Interface

2003 December

Contents 2003 December

[表紙デザイン：㈱プランニング・ロケッツ]

特集

戦略的な再利用で魅力的な商品を次々と生み出す!

Interface

Contents 2003 December

話題のテクノロジ解説

ショウレポート&コラム

一般解説&連載

■情報のページ

連載「組み込みGUI設計の現状とソリューション」,「SDIOカード開発入門」,「フリーソフトウェア徹底活用講座」,「開発環境探訪」,「シニアエンジニアの技術草子」は,お休みさせていただきます.

アジア最大のディジタル総合展

WPC EXPO 2003

北村俊之

「～ビジネスが拡がる．生活が変わる．～実践ユビキタス・ネットワーキング」をテーマに「WPC EXPO 2003」が9月17日(水)～20日(土)の4日間，幕張メッセで開催された．主催は日経BP社．延べ来場者数は，昨年をやや下回る263,205人となっていた．また，昨年と比較すると，台湾や中国，韓国などアジア近隣諸国からの出展が目立っていた．

今年は地上デジタル放送パビリオンやインターネットITS/テレマティクスパビリオンなどを設けた「ホーム＆パーソナルゾーン」，ユビキタスIDセンターパビリオン，バイオメトリクス認証体験コーナーなどの「ビジネスゾーン」が新設された．

この種の展示会では恒例となった感がある，携帯電話やディジタルカメラ，カラープリンタ，DVDレコーダなどのコンシューマ向け製品が展示の主流となっており，相変わらず盛況だった．反面，ここ数年のソフトウェア業界の低迷を反映してか，例年にも増してソフトウェアメーカーの出展が減少していた．また，マイクロソフトをはじめとするいくつかのブースでは，8月に猛威をふるったコンピュータウィルスに対応するため，同社が提供する「Windows XPセキュリティ対策」CD-ROMの配布も行われていた．

● 低迷するソフトウェア業界

ここ数年，ソフトウェア業界の低迷，とくにパッケージソフトの低迷が叫ばれている．こうした状況を反映してか，今年は例年になくソフトウェアメーカーの出展が少なかったように感じられた．

そうした中で来場者の高い注目を集めていたのがエッジである．同社は，8月末に発売されたばかりの，LinuxベースのOS「LindowsOS 4.0日本語版」(開発：米Lindows.com)と対応ソフトウェア/ハードウェアを多数展示していた(**写真1**)．低コストや信頼性，安全性といった，同OSのメリットをアピールするメインステージに多くの来場者が足を止めていた．今回は，DDIポケットのAirH"への対応予定を表明したことでも話題となった．

〔写真1〕
LindowsOS4.0日本語版を展示するエッジ

マイクロソフトでは，10月に発売予定の「Microsoft Office 2003」をメインに展示を行っていた．また，パートナー企業各社による「Microsoft Windows 2003 Server」を利用したソリューションも展示されていた．そうした中でオズプロジェクトは，ファミリーレストランなどでの利用に最適なオーダリングシステムの展示デモを行っていた(**写真2**)．こちらのシステムでは，同社の「Server Based Computing-POS」技術を採用していること，従来のPOS機能と同等のサービスをYahoo!BBを通じて，より低価格で提供できることなどが特徴であるとのことだった．

〔写真2〕
オズプロジェクトのWindowsベースオーダリングシステム

● 注目の高い携帯電話，ディジタルカメラ

最近のコンシューマ向けの展示会で，来場者の関心がもっとも高いのは，携帯電話とディジタルカメラの新製品の動向ではないだろうか．しかし，時期的なこともあるのだろうが，新製品を展示しているブースが意外と少なかったのが実情である．

ボーダフォンへのブランド変更を間近に控えたJ-フォン(**写真3**)では，ボーダフォンブランドの第1弾製品となる「V801」の展示を行っていた．同製品はボーダフォンライブに対応し，海外でも国内同様メールやWeb，Vアプリが利用できることを大きな特徴としている．また，TVコールにも対応しており，同機能を搭載している機種同士では，相手の顔を見ながら会話が楽しめるとのことである．

〔写真3〕
J-フォン(ボーダフォン)のブース

キヤノン，ソニー，ニコン，東芝，三洋電機，カシオ計算機などのそれぞれのブースでディジタルカメラが展示されており，いずれのブースでも実機を手にとって試してみようという来場者でにぎわっていた．ソニーブースでは，同社が16日に発表したサイバーショットシリーズの最上位モデル「DSC-F828」(**写真4**)が展示されており，12月中旬予定の発売日の前に実機を手にしようという来場者が殺到していた．同製品は，「カールツァイス バリオゾナーT*」レンズを搭載しており，光学7倍ズーム(最大14倍プレシジョンディジタルズーム)，RGBにエメラルドフィルタ(E)を追加した「4 color Super HAD CCD」の採用などを特徴としている．

〔写真4〕
サイバーショットシリーズの最上位モデル「DSC-F828」

三洋電機は，ムービーデジカメ「DSC-J2」を全面に打ち出したブース構成をとっていた．同製品は320万画素のCCDを搭載し，静止画で約600万画素相当，動画で最大640×480ドット(30フレーム/s)の性能をもっている．記録メディアはSDメモリカード/マルチメディアカード(MMC)を利用．また，発売日未定のディジタルフォトプリンタも参考出品されていた．

● 参考出品に未来が見える

新製品という面では，期待できる部分はあまり多いとはいえなかったが，参考出品レベルでは今後期待がもてそうな技術がいくつか見られた．東芝ブースでは，MPEG-4デバイスのコンセプトモデル(**写真5**)がいくつか展示されていた．B6サイズの本体に，無線LANアダプタを接続するPCカードスロットと動画を記録するSDメモリカードスロット，TVチューナおよびMPEG-4変換ユニット(CODEC)を内蔵した「MPEG4応用パーソナルビデオサーバ」は，アンテナをつなぐだけでTV番組の録画，無線LAN経由での配信が可能だという．またそのモニタとして小型ディスプレイ「MPEG4 Mobile Viewer」も展示されていた．MPEG-4の再生機能と無線LAN/Bluetooth対応の無線通信機能，1.8インチサイズで20Gバイトのハード，VGAサイズで記録可能なCMOSカメラとVGA表示の4インチ低温ポリシリコンTFT液晶ディスプレイを搭載している．

タカラでは，Sバンド衛星ディジタル放送用の携帯型受信端末のコンセプトモデルの展示を行っていた(**写真6**)．現在のバッテリのもち時間は約1.5時間．往復の平均通勤時間(片道約70分)程度をクリアすることを目標にしているという．

〔写真5〕
東芝のMPEG-4デバイス

〔写真6〕
タカラの携帯型受信端末

自動認識技術の総合展示会

第5回 自動認識総合展

北村俊之

「HUMAN&AUTO-ID ――生活と産業を支える自動認識技術」をテーマに「第5回 自動認識総合展」が9月10日(水)～12日(金)の3日間，東京ビッグサイトで開催された(**写真1**)．主催は(社)日本自動認識システム協会．今回で第5回目を迎える同展示会は，バーコード，2次元シンボル，RFID，カード(IC，磁気ほか)，バイオメトリクスおよびシステムなど，自動認識技術に関する幅広い展示会となっていた．昨年に引き続き，バイオメトリクスとカードに関しては「BIOMETRICS EXPO」および「CARD EXPO」として併催されていた．出展社158社，団体428小間と，昨年を上回る過去最大規模での開催となり，三日間の来場者数は24,606人であった．

〔写真1〕入場口のようす

● AUTO-ID EXPO

サトーでは，ICタグ，2次元コード，バーコードを活用したソリューションを多数展示していた．いつでも，どこでもその場でラベル発行，ID発行を実現するユビキタスマーキングの具体事例に来場者の関心も高かった．とくに今回新登場の「TASSHA」(**写真2**)は，防滴，防油，抗菌，害虫忌避，省エネなどの特徴をもつ．小型ながらも高速で鮮明な印刷，豊かな表現力をもっており，同社でもスタンドアロンプリンタの未来形と位置づけるなど，現在もっとも注目度の高い製品であるという．

〔写真2〕サトーのスタンドアロンプリンタ「TASSHA」

オムロンは，2次元コードリーダおよびRFIDを中心とした展示を行っており，製造現場から流通分野まで幅広い分野をカバーしていることをアピールしていた．ダイレクトマーキング対応のハンドヘルドタイプの2次元コードリーダ「V530-H3」(**写真3**)や固定型2次元コードリーダ「V530-R2000」などは，来場者の関心の高い製品であるとのことだった．

〔写真3〕オムロンの2次元コードリーダ「V530-H3」

日本シンボルテクノロジーでは，小型オムニプロジェクションスキャナ「LS9208」をはじめ，バーコードレーザスキャナの新製品が多数展示されていた．ターミナルでは，Windows CE .NETを搭載したインダストリアルPDA「PPT8846」に加え，イメージリーダ搭載IEEE802.11b準拠無線LAN対応「Pocket PC PPT8846」の参考出品が行われていた．また同ブースでは新しいワイヤレスLANのあり方を定義した「Symbol Wireless System」が展示されていた．こちらはオープンで拡張性の高いアーキテクチャを提供，無線LANパフォーマンスの向上，セキュリティの強化，メディアの独立性，TCO低減の実現などを特徴としており，来場者の関心が高いソリューションであるとのことだった．

シージェーピーテクノロジーは，バーコードラベルの分散，現場発行に適する小型デスクトッププリンタおよび携帯型プリンタを多数展示していた．また今回新製品として，テクノベインズ製の防塵対応ファンレスコンピュータ「UCM0822A」の展示も行われていた．これはファンレスコンピュータとしては，業界初の密閉型の防塵対応を実現しているとのことだった．

松下電器産業では，携帯電話との接続を実現した，バーコードリーダ(**写真4**)が来場者の注目を集めていた．

〔写真4〕松下電器産業の携帯電話との接続を実現したバーコードリーダ

● BIOMETRICS EXPO & CARD EXPO

今回のBIOMETRICS EXPO & CARD EXPOでの最大のテーマは，「本人認証」である．バイオメトリックスでの本人認証といえば，指紋による認証が一般的だったが，現在では目や耳，顔から掌，静脈まで，身体のあらゆる部分が認証の対象となっている．

シーベルは，虹彩による本人認証入退出管理システム「Iris Access3000」(**写真5**)を中心に展示デモを行っていた．同社では，この虹彩による本人認証が，現在もっとも高信頼性なシステムであると考えているとのことである．

〔写真5〕シーベルの虹彩による本人認証入退出管理システム「Iris Access 3000」

シンクロでは，静脈パターン認証システム「VP-Ⅱ」(**写真6**)の展示デモで来場者の関心を集めていた．同システムは，手の甲の静脈パターンを近赤外線光学システムで認識し，最新のアルゴリズムでパターンマッチングを行うことで個人認証を行う．他人受入率0.0001%，本人拒否率0.1%，認証速度0.4秒と優れた性能を維持しているのが大きな特徴とのことだった．エム・エー・ジェーでは，掌形判別器「ハンドキー(HK-2)」の展示デモを行っていた．上方からCCDカメラで掌の像をとらえ，指の長さと幅のデータを，側面から手の厚みのデータを測定し，それらの情報をディジタル信号に変換してメモリに保管するというものである．1秒以内で認証結果がディスプレイに表示されるという．東芝は，顔照合セキュリティシステム「FacePass」の展示デモを行っていた．同システムは，セキュアでありながらユーザビリティに優れたシステムをめざしているという．

〔写真6〕シンクロの静脈パターン認証システム「VP-Ⅱ」

アマノでは，就業管理，入出管理に関する問題を解決する幅広いソリューションの展示デモを行っていた．とくに，非接触ICカードや指紋認証技術を応用した出退勤データ収集リーダの就業情報ターミナル(**写真7**)，入退室データ入力，開錠のための入室情報ターミナルなどは人気で，力を入れているラインナップであるという．

アイアンドディは，顔写真入りIDカード発行システムの展示を行っていた．こちらは新製品である「撮影BOX」とIDカード発行ソフト「ID Maker」を組み合わせることで，誰でも顔写真入りのIDカードを簡単に発行できるという．とくに「ID Maker」はワープロ感覚で簡単に操作ができ，磁気カード，バーコード，ICカード(Mifare，Felica，TN2など)など多彩なカードに対応していることが特徴だという．幸和システム販売は，「ファーゴ カードジェット410プリンタ」(**写真8**)の展示を行っていた．発行ソフト，背景テンプレート，バーコードを用いたアプリケーションテンプレートソフトの3種類がバンドルされたモデルでは，低価格で高品質なIDカードの作成が可能であるという．

〔写真7〕アマノの就業情報ターミナル

〔写真8〕幸和システム販売の「ファーゴ カードジェット410プリンタ」

Show&News Digest

T-Engineフォーラムに マイクロソフトが参加

■日時：2003年9月25日（木）
■場所：都ホテル東京（東京都港区）

T-Engineフォーラムとマイクロソフト（株）は共同で記者会見を行い，マイクロソフト社がT-Engineフォーラムに幹事会員として参加し，T-Engineプラットホーム上でWindows CE .NETを動作させるための仕様策定などを共同で進めることを発表した．発表会にはT-Engineフォーラム会長で東京大学教授の坂村 健氏と，マイクロソフトのバイスプレジデント古川 享氏が出席した．

坂村氏は「（T-Kernelなど）リアルタイムカーネルはマイクロ秒を争う応答を扱うものであり，一方（Windowsなど）情報系OSはユーザーインターフェースなどを中心とした数ミリ秒単位の応答を扱うOSだ．相手にしているものや用途が違う」と，それぞれが競合するものではないことを説明した．T-KernelとWindows CE .NETの役割は明確だとして，その一例にディジタルカメラをあげ，「たとえばオートフォーカスなど動作スピードが要求されるところはT-Kernelで行い，撮影した画像をインターネット経由で送信したりする部分はWindowsで行う」と説明した．「車のエアバッグが出るときに砂時計が出てきたら嫌でしょ？」と，古川氏からジョークの利いた発言も飛び出した．

T-Engineフォーラム会長の坂村 健氏（左）とマイクロソフト社の古川 享氏（右）

二つのOSのT-Engine上への実装方法は，割り込みやスケジューリングをT-Kernelで行い，資源管理を二つのOSで分割する「部分ハイブリッド型」を採用する．また将来的には，T-Kernel上にタスクとしてWindows CE .NETを載せる「カーネルオンカーネル型」の採用もありえるという．この点に関して坂村氏は「Windows CE .NETは，T-Kernelから見れば巨大なミドルウェアの一つ」と表現している．

具体的な開発成果に関して古川氏は「TRONSHOWで実際に動くものを見せたい」としている．TRONSHOW 2004は，2003年12月11日（木）～13日（土）に東京国際フォーラム（東京都千代田区）で開催される．

NPO法人TOPPERSプロジェクト 第1回通常総会・設立記念講演会

■日時：2003年9月12日（金）
■場所：虎ノ門パストラル（東京都港区）

オープンソースのITRON実装系であるTOPPERSが8月にNPO法人として認可され，総会と設立記念講演会が開催された．

講演会では，会長の高田広章氏によりTOPPERSプロジェクトの進捗状況が語られ，なかでも苫小牧高専で開発されたTCP/IPプロトコルスタック「TINET」が注目を集めていた．

経済産業省情報処理振興課の田代秀一氏による招待講演「オープンソースソフトウェアへの期待」では，「中身がわかっているOSのほうが高信頼性ソフトウェアを作成しやすいのではないか」，「セキュリティ（バグの少なさ，検証可能性：バックドア問題），特定ベンダへの非依存，国内技術・産業の育成という観点からも，（TOPPERSのような）オープンソースソフトウェアには意義がある」と，同プロジェクトに対する経産省の期待を明らかにしていた．

経済産業省情報処理振興課の田代秀一氏

QNX技術セミナー

■日時：2003年9月18日（木）
■場所：カナダ大使館（東京都港区）

QNXソフトウエアシステムズ（株）と（株）ルネサス テクノロジにより「パワーマネジメントとウェブサービス：モバイルデバイスでの電力消費とサービスを最適化する新しいフレームワーク」と題したセミナーが開催された．

従来のBIOSまたはOSによる電力管理では，組み込み製品でのニーズに対応できないとの観点から，QNXではあらたに「QNXパワーマネジメントコンポーネント」「パワーフレームワークAPI」を提案し，アプリケーションによるきめ細かな電力管理を可能にしたとのことである．

SystemCデザイン・ワークショップ＆SystemVerilogデザイン・ワークショップ

■日時：2003年8月29日（金）
■場所：パシフィコ横浜（神奈川県横浜市）

CQ出版社の主催により，SystemCとSystemVerilogに関するワークショップが開催された．従来のHDLによるハードウェア設計から，SystemCなどによるシステム設計への移行期ということもあり，多くの入場者を集めていた．行われたチュートリアルは，富士通（株）長谷川隆氏による「SystemCの標準化最新動向」，日本シノプシス（株）西園寺修氏の「SystemCによるシミュレーション」など．

ハッカーの常識的見聞録

36

広畑由紀夫

これから発売されるデスクトップ/ノート向け 32ビットプロセッサアーキテクチャについて

☆ 今回は，これから実装されるインテル系プロセッサアーキテクチャについて，その特徴や利点をまとめて，今後の導入の参考にしていただければと思います．

● Pentium4 Extreme Edition 3.2GHz

Pentium4 Extreme Editionとは，2003年9月16日，急きょ発表されたPentium4のハイエンド製品です．すでに発表されているPrescotteコアとはことなり，現行のNorthwood-HTコアにXeon-MPで実装されているL3キャッシュを2Mバイト実装したデスクトップPC向けPentium4です．現行のNorthwoodコアと同じく0.13μmで製造されるため，90nmで製造されるPrescotteまでの中繋ぎ製品であると思われますが，価格によっては大量に採用される可能性の高いプロセッサといえそうです．

また，公式にはデュアルプロセッサマザーボードこそ販売されていませんが，ASUS社などからPentium4向けFSB800MHz仕様のデュアルプロセッサマザーボードなどが発売されていることもあり，家庭用デスクトップPCとしては最高性能を引き出すのではないかと推測されます．

● 今後期待されるアーキテクチャ(1)「Vanderpool」

マイクロソフトがConnectix社の仮想マシンテクノロジを買収したように，近年コンピュータの仮想化に拍車がかかっているようです．また，マイクロソフトはこうした仮想コンピュータサーバの発売の計画もあるとのことです．

こうしたソフトウェア上の仮想コンピュータは複数のハードウェアで構成されるシステムから見た場合，見かけの台数は増やせてもパフォーマンスにおいては著しく低下することは容易に想像できることでしょう．実際に，Connectix Virtual PCシリーズを導入し仮想コンピューティングを行っている人にはメインのPCにかなりの負担が生じ，実際の動作速度にかなりの影響が出ると感じていることと思います．

「Vanderpool」アーキテクチャでは，CPU自体に複数の仮想コンピュータ実行機能をもたせることで，マルチスレッドやハイパースレッドを超えたコンピュータの仮想化を実現するものです．その性質上，シングルコアプロセッサでの実現は厳しいと思われるので，次のマルチコアプロセッサに移行してからの実装になると考えられます．

● 今後期待されるアーキテクチャ(2)「マルチコア」

Vanderpoolアーキテクチャのようにさらに多くの処理を行うには，シングルプロセッサでの実現はどうしても限界があります．そのため，複数のコアをもち，プロセッサ自体をシングルコアプロセッサの個数倍の処理能力をもたせることで，Vanderpoolアーキテクチャやさらに高度なプロセッシングを実行することが考えられます．

Pentium4およびXeonシリーズでは，今後2個のコアを実装したデュアルコアが計画され，Itanium2の後継版ではさらに多くのマルチコアが計画されているようです．いずれこうしたサーバ用アーキテクチャもデスクトップPC向けに実装されてくることでしょう．

● ノートPCに期待されるテクノロジ

デスクトップを置くスペースはないが性能は欲しいという人に，Hyper-Threading対応のPentium4-Mの導入予定は朗報だと思います．長時間バッテリ駆動可能なノートPCも非常に便利ですが，デスクトップPCのように使用されるデスクノートクラスでは，長時間バッテリ駆動よりパフォーマンスが必要な場合が多いことでしょう．ノートPCにHyper-Threadingが実装されたプロセッサが採用されることで，従来はデスクトップでしかできなかった作業の多くが，ノートPCでも快適に作業できる環境が整ってくると思われます．

● 今後の期待

マルチコアアーキテクチャおよびVanderpoolアーキテクチャの実装で，1台のPCが複数のPCをよりよく兼ねるハードウェア仮想PCが家庭内でも安価に入手できるようになることでしょう．ハードウェア仮想PCにより，物理的に1台のPCを2台として使えるようになると，1台のPCでクラスタリングやフェイルオーバなどを実装することが可能になり，サーバのメンテナンスや24時間駆動サーバが，より身近になってくることと思われます．

ノートPCにおいてもチップセットとの組み合わせで，より省電力駆動のモデルやHyper-Threadingを使用した，全体としてより高速に動作するモデルなどの幅も広がってくることでしょう．今，筆者が導入予定しているのはもちろん，Pentium4 Extreme Editionです．

- Pentium 4 Extreme Edition リリース情報(英語)
  http://www.intel.com/pressroom/archive/releases/20030916corp_b.htm
- Vanderpool インフォメーション(英語)
  http://www.intel.com/ca/pressroom/2003/0916.htm

ひろはた・ゆきお　OpenLab.

特集 戦略的な再利用で魅力的な商品を次々と生み出す！

具体例で学ぶ 組み込みソフトの再利用技術

組み込み機器開発は，日本の「お家芸」である．しかし最近の傾向として，世界的に組み込み機器への関心が高まり，これまで組み込みを経験したことのない技術者が組み込み機器を開発し，ワールドワイドな市場で製品展開を行おうとしている現状がある．組み込みLinuxやWindows CE .NETの展開も，その状況を後押しする要素となっている．一方，日本の組み込み機器開発の現場では，市場からの要求は多様化しているのに，開発サイクルまで短くなるという，つらい状況が続いている．

そこで本特集では，実際の組み込み機器開発で得た成功体験をもとに，組み込み機器の特徴を分析し，体系的な組み込みソフトの再利用技術で問題解決を行うというアプローチを，できる限り具体的な例を使い，実装例まで含めて，ていねいに解説する．

プロローグ　体系的な再利用を実現するプロセス

魅力ある製品を生み出すためのソフトウェア開発とは?

酒井由夫

はじめに

本特集記事は，組み込みシステム商品群におけるソフトウェア開発プロジェクトのリアルなシミュレーションです．現場のソフトウェアエンジニアが組み込み商品群のソフトウェアを開発する際に，常に納期に追われ「取りあえずやれることをやる」というアプローチによって陥ってしまう悪循環を，***プロダクトライン***という考え方を使って好循環に変え，魅力ある商品を次々に生み出せる体質を作ることが，本特集記事の目的です．

このイメージは，組み込みシステム商品群を開発していく中で作成された，混沌としたソフトウェア資産の固まりを，きれいに整理して再利用可能な単位に分類し，ソフトウェア資産の中でも再利用性が高く商品価値が凝縮されている**コア資産**を摘出し，摘出したコア資産を使ってさまざまなバリエーションをもたせることで，商品ラインナップをそろえていくというものです(**図1**，**図2**)．

体系的な再利用を行うのは，ソフトウェアの資産だけではありません．商品開発のプロセスや開発環境，ドキュメント，テスト手法など同じ製品群で繰り返し使う知識や手法なども，再利用の単位になります．

1 プロセスの進め方

体系的な再利用を使った組み込み機器の商品開発をシミュレーションするにあたって，次のような章立て・ストーリを考えました．

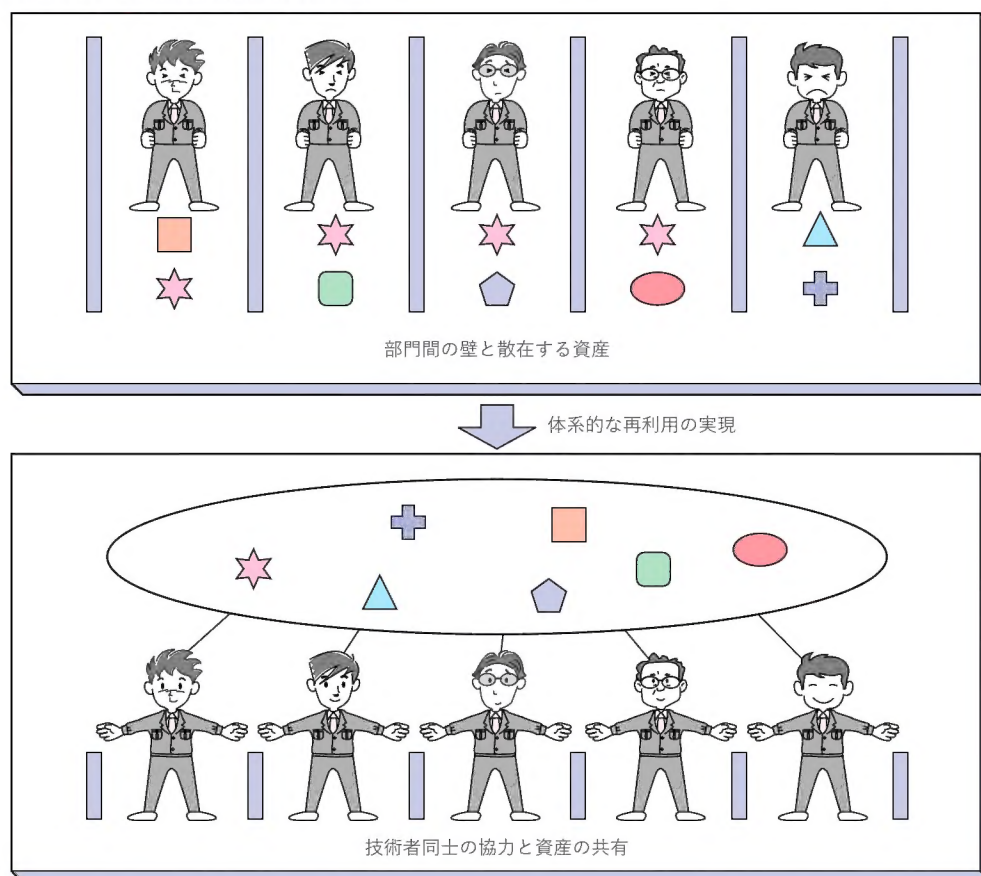
- **第1章**：組み込みシステムを分析し，対象となる組み込み商品の特徴を把握する
- **第2章**：体系的な再利用とは何かを，***プロダクトライン***という考え方を通して解説し，対象となる商品群のいろいろなロードマップを描いてみる
- **第3章**：商品群のロードマップをもとに，要求仕様から機能を洗い出し，再利用可能な機能ブロックを単位としたドメイン構造図を作成する
- **第4章**：ドメイン構造図からコア資産と成り得るドメインを摘出し，その中身を実装する
- **第5章**：組み込みシステムに対する安全性・信頼性を分析し，どのようにテストするかを考える

本特集記事の工程の流れ(プロセスフロー)，各工程(プロセス)からのアウトプット，体系的な再利用活動の関係については，**図3**を見てください．

〔図1〕体系的な再利用の実現

〔図2〕ラインナップの実現

〔図3〕特集記事のマッピング

〔図4〕部門間の壁を引き下げ，体系的な再利用を実現する

2 オブジェクト指向設計の必要性について

第4章では，摘出したコア資産の実装について，**UML**(**Unified Modeling Language**)を使って分析・設計を行い，オブジェクト指向言語である**C++**を使ってコードを作成しています．しかし，**体系的な再利用を行うにあたって，UMLやC++の使用が絶対条件になっているわけではありません．**

第1章で組み込みシステムの特徴を把握し，第2章でロードマップを描き，第3章でコア資産を摘出し，第5章でシステムの安全性・信頼性を分析・検証することができれば，実装は従来のプログラミング手法を使ったとしても，体系的な再利用は実現可能であり，このような取り組みを行っていなかったときから比べると，はるかに効率のよい商品開発ができるようになるはずです．

3 どのような組み込み機器にも使えるのか

体系的な再利用は，どんな組み込みシステム開発にも生かすことは可能ですが，それぞれの領域・分野に合わせてアレンジする必要はあります．本稿では，このようなアレンジを実施しやすいように，**電子ポット商品群**というリアルな具体例を示し，この具体例をいろいろな角度でながめ分析することによって，抽象的概念への転換がしやすいような構成をとっています．よって，家電商品群以外の組み込み機器業務ドメインへの投影・適用も可能です．

4 組織としての取り組み

体系的な再利用を実現するためには，組織としての取り組みも重要です．部門間の障壁を引き下げ，それぞれの部門が所有している資産の中で共有できるものを摘出してマネージメントすることが，組織全体の開発効率を上げることに貢献します(**図4**)．

さあ，読者のみなさんも，体系的な再利用による組み込み商品群のソフトウェア開発シミュレーションを体験してみてください．

さかい・よしお　組込みソフトウェア管理者・技術者育成研究会(SESSAME)

第1章　組み込みシステムの特徴の理解と，問題解決の道筋探索

組み込みシステムの特徴を考える

酒井由夫/稲葉道夫

「組み込みシステム」と一口にいっても，多種多様な種類があり，続々と新たなシステムも誕生している．また組み込みシステムは，人によってその定義や認識が変わりがちである．そこで本章では，「組み込みシステムとは何か？」を，身のまわりの組み込み機器の列挙からはじめ，その特徴を四つに分けて，さまざまな考察を加え，何が大事かを抽出する．（編集部）

はじめに

私たちの生活の中に，**組み込み機器・組み込みソフトウェア**はいったいどれくらい入り込んでいるのでしょうか？　感覚的には，身のまわりにたくさんあるほとんどの電子機器にはCPUなどのプロセッサが搭載されていて，プロセッサ周辺のデバイスやメカニズムはソフトウェアで制御されている感じがします．そこで，組み込み機器とは何かを具体的に認識するために，家庭の中にある，CPUが搭載されている組み込み機器をカウントしてみました(**表1**)．

表1にあげたものは家庭の中で使われている組み込み機器ですが，実際には家の外でも自動車やバイク，電車，信号や踏切など交通関連の機器や，自動販売機，キャッシュディスペンサーなどの無人端末，自動ドア，工場内の設備など，CPUを内蔵しソフトウェアで制御されているさまざまな組み込み機器・組み込み設備が世の中に存在しています．たとえば**図1**のようなイメージです．

図1の中で取り上げられている自動車は，はるか昔は機構メカニズム中心のシステムでした．しかし，いまでは何十個ものCPUが搭載されている組み込みシステムの集合体です．車の基本機能をつかさどるエンジンまわりの部分だけでも，10個以上のCPUが，それぞれの仕事をまかされています．たとえば，

- モータ　：16ビットCPU×5
- ブレーキ：32ビットCPU×4
- バッテリ：16ビットCPU×4
- エンジン：32ビットCPU×1＋16ビットCPU×1

という具合です．ドアミラーのコントロールでさえ，左右1個ずつCPUが使われ，これらの何十個ものCPUが車用のLANでつながれているのです．

1　組み込み機器と組み込みソフトウェアの定義

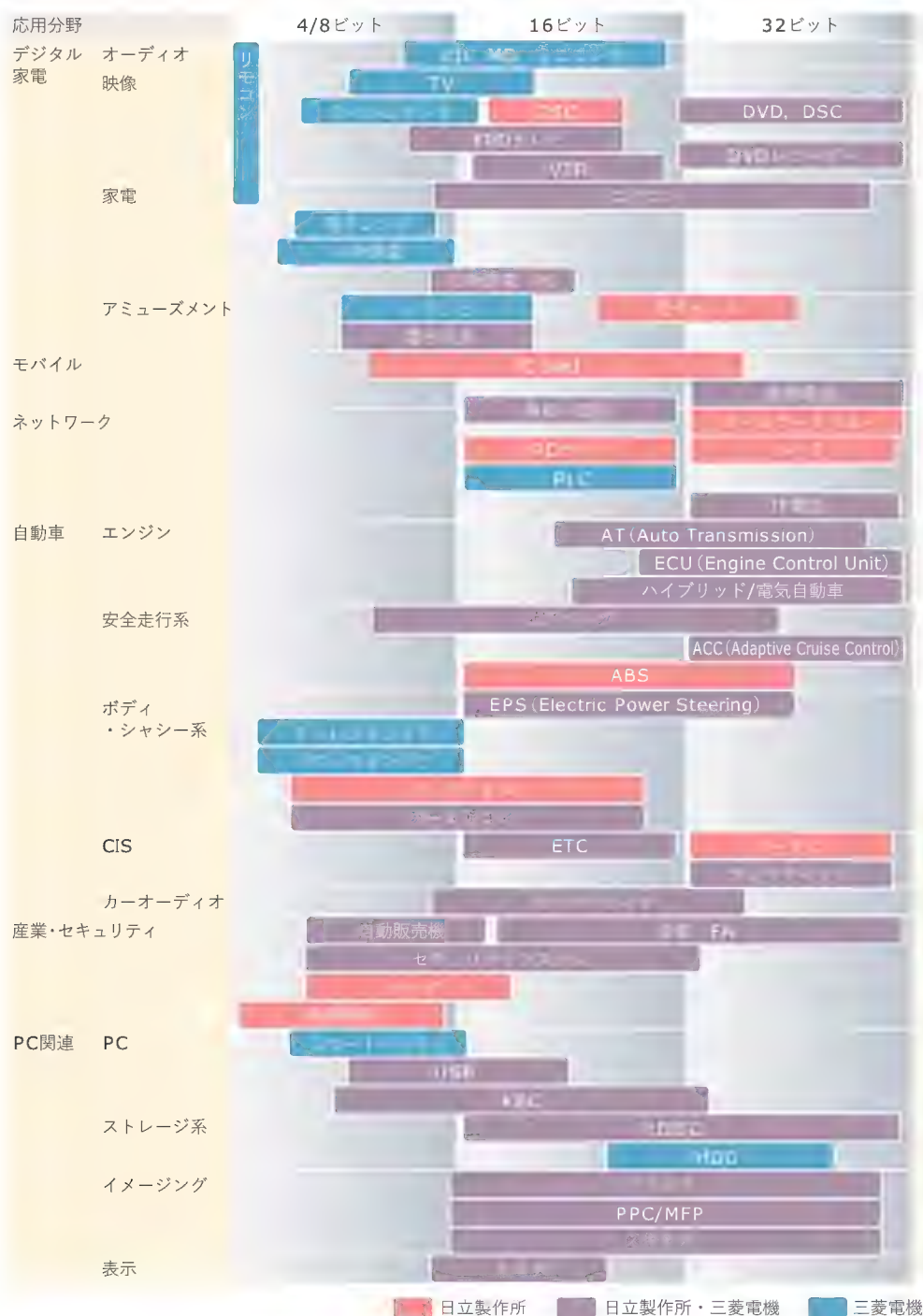
具体的な組み込み機器を列挙したところで，あらためて，組み込み機器と組み込みソフトウェアとは何か？を考えてみます．

CPUが搭載されているにもかかわらず，家庭の中にあって組み込み機器と呼ばれないのは**パソコン**です．パソコンは目的が一つではなく汎用的であり，アプリケーションソフトを入れ替えることなどでさまざまな利用方法を選択できるからです．それとは対照的に，前述の家庭内の組み込み機器には明確な使用目的があります．この二つのCPU内蔵機器の利用形態から，組み込み機器の特徴を考えると，**CPUとソフトウェアが搭載され用途が決まっている機器が組み込み機器であり，CPUは搭載されているけれども，ソフトウェアは入れ替えが可能で目的が一つではない機器は組み込みではない**ということになります．

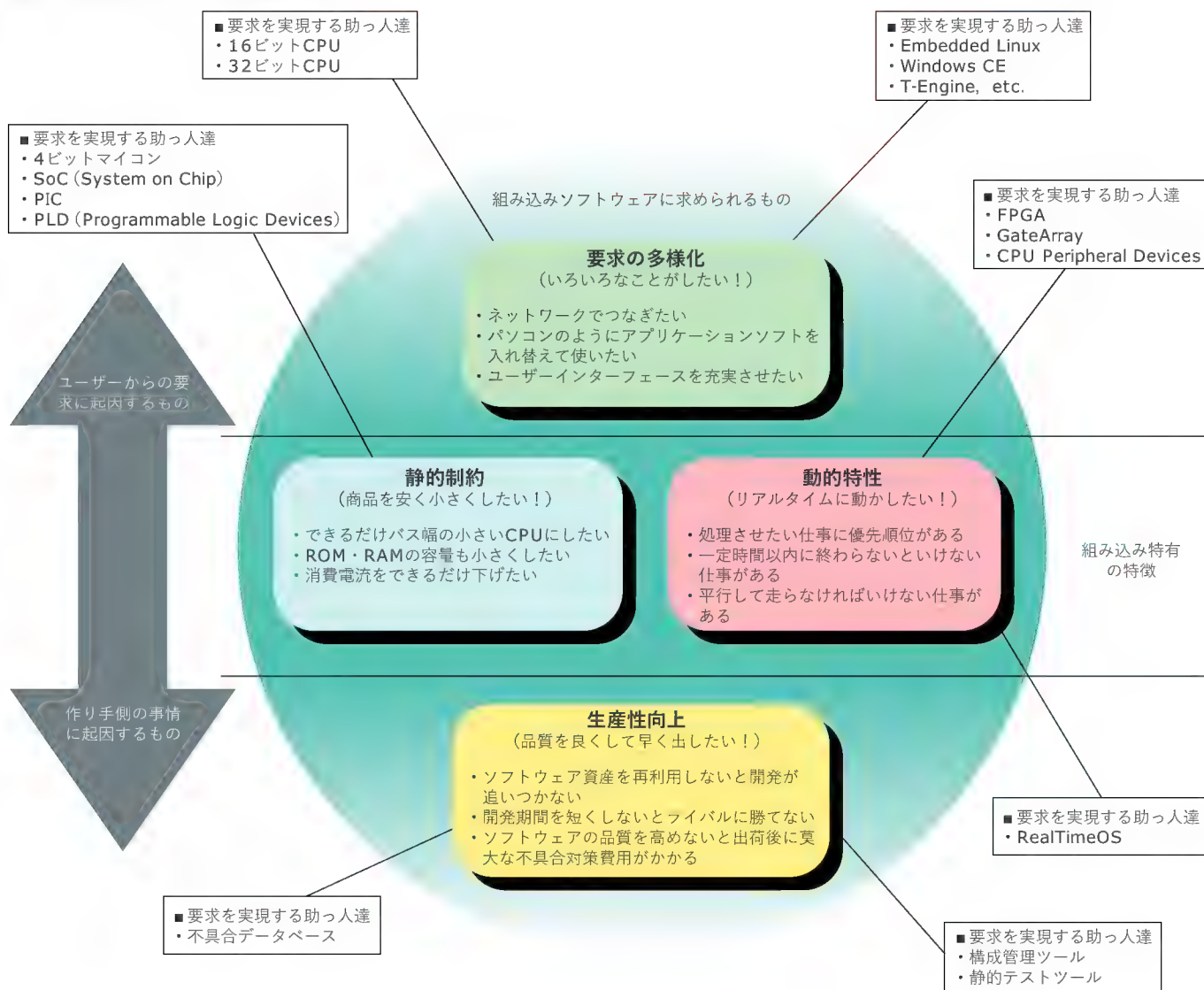
しかし，たとえば最近の携帯電話では，iモードでインターネットにアクセスしたり，iアプリのでゲームをダウンロードし

〔**表1**〕家庭内のCPU搭載組み込み機器

1	テレビ(テレビ本体＋テレビのリモコン)
2	ビデオ(ビデオ本体＋ビデオのリモコン)
3	CSチューナ(CSチューナ本体＋チューナ本体のリモコン)
4	石油ファンヒータ
5	エアコン(エアコン本体＋リモコン)
6	FAX付き電話
7	FAX電話の子機
8	ビデオカメラ
9	携帯電話
10	ミニコンポ
11	食器洗い乾燥機
12	扇風機
13	電子レンジ
14	冷蔵庫
15	ガスレンジ
16	給湯器
17	電子ポット
18	炊飯器
19	デジカメ
20	電波時計
21	電卓
22	プリンタ
23	家庭用ミシン

〔図1〕さまざまな組み込み向けCPUの例：「ルネサスマイコンのソリューション別マップ」〔(株)ルネサステクノロジ『RENESAS EDGE』創刊号より〕

遊んだりできます．携帯電話はCPUとソフトウェアが搭載され，電話をかけたりメールを交換したりするという目的がある組み込み機器の代表格のようなものですが，Web閲覧やいろいろなゲームを行うこともできます．どうも，組み込みか組み込みでないかは，単純にソフトウェアが入れ替えられるかどうかだけでは判断できそうにもありません．

そこで，さまざまな用途で使われる組み込み機器の特徴を分析し，組み込み機器の本質を捉えるため，組み込みシステム・組み込みソフトウェアを四つの視点で分類し，ながめてみることにします．

New Products ──スイッチング周波数が1.3MHzの昇圧型スイッチングレギュレータIC「LT3460」
リニアテクノロジー(株)は，昇圧型スイッチングレギュレータIC「LT3460」を発売した．スイッチング周波数1.3MHz，入力電圧2.5〜16V，出力電圧36V$_{max}$である．また300mA対応のスイッチ素子を内蔵する．パッケージは6ピンのSC70/5ピンのThinSOTである．

〔図2〕組み込みソフトウェアの特徴(1)

2 組み込みシステム・組み込みソフトウェアの特徴を分析する

組み込みシステム・組み込みソフトウェアを四つの特徴で分類すると，**図2**のようになります．

1. **要求の多様化**(いろいろなことがしたい!)
2. **静的制約**(商品を安く小さくしたい!)
3. **動的特性**(リアルタイムに動かしたい!)
4. **生産性向上**(品質を向上し早く商品を世の中に出したい!)

上記について，一つ一つ説明します．

要求の多様化とは，生活が豊かになってモノがあふれてきた現代社会の中にあって，人々がより便利なものを求めるようになった結果，顕著になってきた特徴です．たとえば，固定電話に留守録機能が付き，コードレスの子機が付き，アドレス帳が付き，FAXが付き，ナンバーディスプレイ機能が付き，コピーまで取れるようになりました．人間の欲求は，とどまるところを知りません．このような要求の多様化に対応するため，組み込みソフトウェアには柔軟性が要求されます．

静的制約は，おもに小さくしたい/軽くしたいという要求からくることが多いのですが，安くしたいという要求も，静的制約の大きな要因の一つです．携帯型の機器で小さく，軽くするには当然バッテリも小さくする必要があり，電力をできるだけ消費しないデバイス群で機器を構成する必要があります．どんな組み込み機器でも，大きくて重いほうがよいものはそうありませんから，組み込み機器に静的制約はつきものです．

動的特性が要求される理由は，人々の生活の中にその答えがあります．人は連続した時間の流れの中で生きているので，人が接

する機器にも連続的かつ敏捷に反応してほしいと考えます．スイッチを押してからレスポンスが返ってくるまで数十秒もかかるような組み込み機器は，とうてい顧客に受け入れられません．人間は組み込み機器に対して擬人的な感覚をもっており，組み込み機器はオペレータの意図をくみ取り目的を果たしてくれる「部下」のようなものだと考えています．何度説明しても（オペレートしても）思いどおりに動かず，おまけに処理能力が低い部下を使いたくないのは，生身の人間でも組み込み機器の場合でも同じです．

生産性向上という意味は二通りあります．一つは，作り手側から見た商品の品質という意味で，できるだけ設計したとおりに安定した商品を世に出したいということと，もう一つは使う側からの見方で，信頼性や安全性の高い商品を購入したいという意味です．作り手側から見た生産性の向上は，使い手側から見ると商品の安全性や信頼性の確保となります（**表2**）．

3 この分類をいろいろな組み込み機器に当てはめてみる

前述した組み込み機器の四つの特徴を，いろいろな組み込み機器にあてはめてみます．四つの特徴の一つ一つについて，5段階の評価をつけて分類してみました（**図3**）[注1]．

プリンタなどの事務機器の場合，FAXやスキャナ機能を内蔵させたいなどのさまざまな要求があります．したがって要求の多様化のポイントは高く，モータ制御やトナーの塗布などを高速かつリアルタイムに行わなければならないので動的特性も高いのですが，静的制約はそれほどでもなく，生産性向上の必要性が高い組み込み機器といえます．

それとは対照的に，4ビットのCPUを使うような汎用的な目覚まし時計では要求の多様化はほとんどなく，小型化し，電池をもたせたいので静的制約のポイントは高くなります．また動的必要性はそれほどでもなく単純な機能であり，生産性向上の必要はそこそこで済みます．

また，携帯電話はパソコン並みの多様性が要求されるので要求の多様化のポイントが高く，早いレスポンスが必要で，バッテリの寿命を長くするために消費電力をおさえて，次々と短期間に新機種をリリースする必要があるので，動的特性も静的制約も生産性向上の必要性も大きいといえます．

ただし，同じデジタル腕時計のような商品群であっても，目標価格1,980円くらいの普及型の機種では要求の多様化が軽く，静的制約が重い傾向があり，目標価格23,000円くらいのハイエンド機では，要求の多様化が重く，静的制約が軽い傾向があります．要求の多様化を満たすためには，静的制約も少しは目をつぶってくれるということです．もちろん，要求の多様化を満たしつつ静的制約もクリアしているほうが良いに決まっています．

一般的に，単品商品ではなく商品群にグレードがあるような場合は，ハイエンド機種は要求の多様化が重く静的制約は軽くなり，普及型機器は要求の多様化は軽く，静的制約が重い傾向があります（**図4**，p.76）．

しかし，携帯電話などは用途が多様だからといって，商品のサイズが大きくて電池寿命が短いと顧客に買ってもらえません．したがって，携帯電話は要求の多様化も静的制約も，とても重い商品ということになります．**図4**のように天秤が折れるくらい，要求の多様化も静的制約も重いのです．

4 組み込みシステムを支える助っ人達とは？

組み込みシステム・組み込みソフトウェアを実現する際には，いろいろな材料や道具が必要です．材料とはCPUやCPU周辺デバイス，OSのことを，道具とは開発用のツール類のことです．組み込みソフトウェアを実現するための材料・道具を，組み込みソフトウェアを支える助っ人達として，**図2**に情報をマッピングしています．

要求の多様化を支える助っ人達は，16ビットマイコンや32ビットマイコンなどの高性能CPUや組み込みLinuxや，Windows CE，T-Engineなどの高機能OSやハードウェアです．

静的制約を支える助っ人達は，4ビットマイコンやSoC（System on Chip）技術や，PICマイコン，PLD（Programmable

〔**表2**〕組み込みソフトウェアの特徴（2）

要求の多様化 （いろいろなことがしたい！）	ユーザーの生活が豊かになっていくと，組み込みシステムへのさまざまな付加的要求が高まる．ただし，用途の多様化に逆行し超小型化にして機能を絞った商品が受け入れられることもある
静的制約 （商品を小さくしたい！）	静的制約（メモリ容量，消費電力など）は，高性能かつ低消費電力のCPUなどの新しいデバイスが登場していくことによって減少していく．この静的制約が減少した分のパフォーマンスを，ソフトウェアエンジニアリングによって生産性向上や用途の多様化の解決に振り分けることができる
動的特性 （リアルタイムに動かしたい！）	人間の生活は実時間の流れで動いているので，生活に密着した組み込みシステムには実時間制限やリアルタイム性に対する必要性が高い
生産性向上 （品質を良くしたい！）	企業間の競争は国を超えて激化しており，高い信頼性が要求されるうえに，商品リリース期間の短縮，開発コストの削減が要求されている．携帯電話のように用途が多様化するのに生産性を向上しなければいけない組み込みシステムもある

注1：**図3**で取り上げた組み込み機器は，一般消費者が購入できる商品に限定している．組み込み機器は，これらのコンシューマプロダクツのほかにもさまざまな機器が存在する（**図1**など参照）．

〔図3〕組み込みシステムの特徴詳細

	目標価格	目標出荷台数（月あたり）	要求の多様化	静的制約	動的特性	生産性向上	組み込みソフトの設計方針	使用するソフトウエアエンジニアリングの例
デジタル腕時計（普及型）	1,980円	10万	3	5	3	4	コスト最優先の商品であり，用途も限られている．従来のソフトウェアエンジニアリング技術で対応可能だが，ハイエンド機との資産の共通化を考えるのであれば再利用の技術を導入する必要がある	・リアルタイムOS
デジタル腕時計（高機能型）	23,000円	1万	4	4	3	4		・リアルタイムOS
電子ポット（普及型）	13,000円	1万	2	5	3	4	普及型の電子ポットは用途が限られているが，ハイエンドの電子ポットも含めた商品群を考えると再利用性を重視したソフトウェアエンジニアリング技術を導入することを検討すべきである	・リアルタイムOS
電子ポット（携帯電話カード付き）	98,000円	5千	5	3	3	4		・リアルタイムOS
電子ポット（ネットワーク対応・携帯電話カード付き）	（月額使用料3,000円）	1千	5	3	3	4		・リアルタイムOS ・オブジェクト指向設計 ・Embedded Linux
電子レンジ	39,800円	1万	5	1	4	4	制御が複雑で，要求の多様化が「重い」ので，再利用性を重視した設計が必要	・リアルタイムOS ・オブジェクト指向設計
レーザープリンタ	46,000円	5万	5	1	5	4		・リアルタイムOS ・オブジェクト指向設計
インクジェットプリンタ	14,800円	10万	4	1	5	4		・リアルタイムOS ・オブジェクト指向設計
ハンディ肩たたき器（強・中・弱の3段階）	2,580円	1万	1	3	3	1	コスト最優先の商品であり，用途も限られている．従来のソフトウェアエンジニアリング技術で対応可能	・組み込みソフト技術者の一般的なスキル
ハンディ肩たたき器（さまざまなもみほぐしパターンあり）	13,000円	5千	4	2	5	3	機構部を複雑かつリアルタイムに制御することが必要．反発力をフィードバックして強さを調節するなどの高度なコントロールが求められる	・リアルタイムOS
目覚まし時計	1,000円	10万	1	5	3	2	コスト最優先の商品であり，用途も限られている．従来のソフトウェアエンジニアリング技術で対応可能	・リアルタイムOS
携帯電話	28,000円	17万	5	5	5	5	再利用性を高める技術を導入する必要がある．技術の導入コストがかかっても回収できる商品群である	・リアルタイムOS ・オブジェクト指向設計
	グレード　5：とても大きい　4：やや大きい　3：ふつう　2：やや小さい　1：小さい							

デジタル腕時計（普及型）

デジタル腕時計（高機能型）

電子ポット（普及型）

電子ポット（安否確認機能付き）

ハンディ肩たたき器（3階段）

ハンディ肩たたき器（複雑なパターン）

〔図4〕要求の多様化と静的制約のバランス

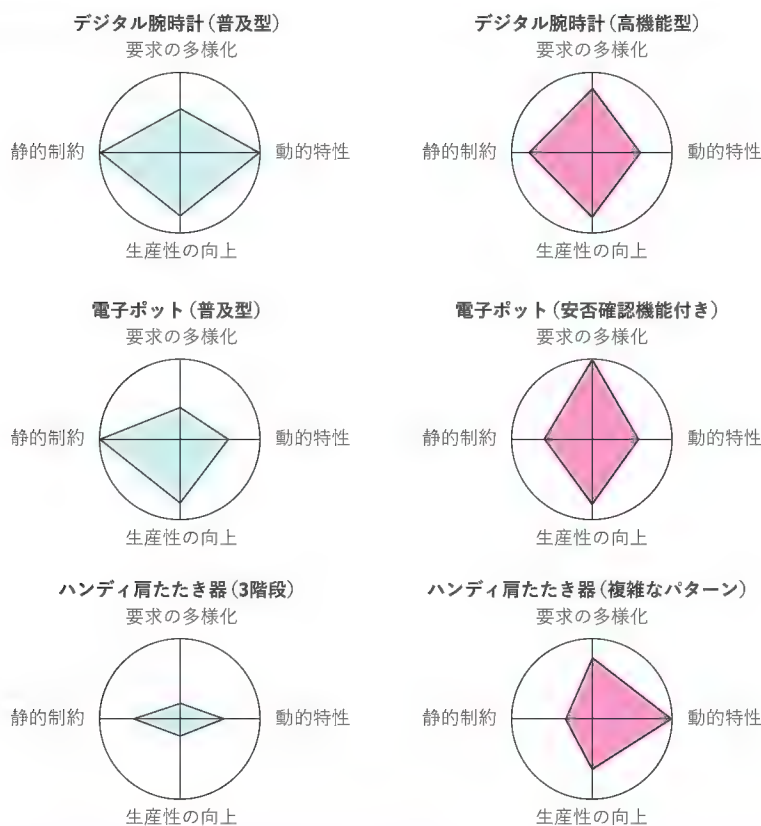
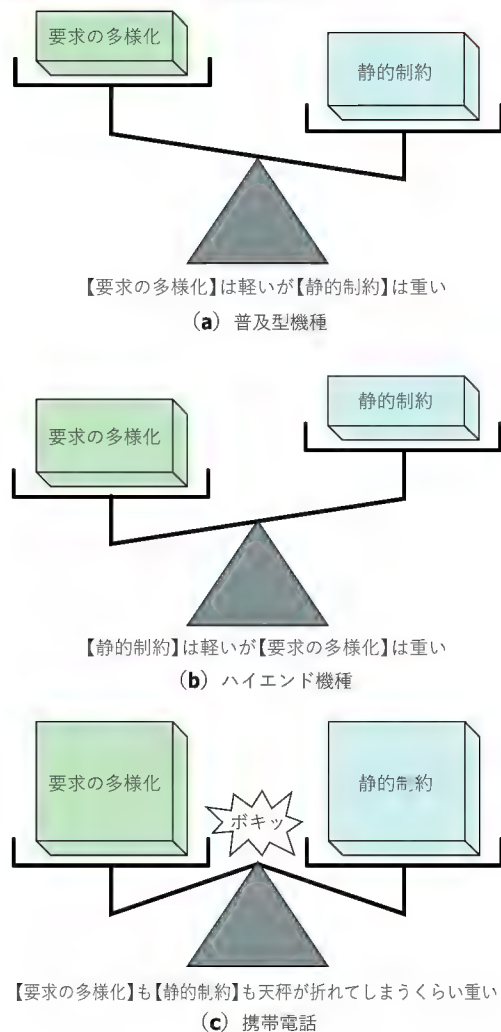
【要求の多様化】は軽いが【静的制約】は重い

(a) 普及型機種

【静的制約】は軽いが【要求の多様化】は重い

(b) ハイエンド機種

【要求の多様化】も【静的制約】も天秤が折れてしまうくらい重い

(c) 携帯電話

Logic Devices)などです．SoC技術により必要な周辺デバイスを同じチップ内に埋め込んだり，PLDで必要なディジタル回路を組み，CPUの負担を軽くしてバス幅の小さいCPUを使ったり，PICマイコンのような安くて処理能力の高いCPUを使うことで静的制約をクリアしていきます．

動的特性を支える助っ人達は，静的制約と同じくPLD(Programmable Logic Devices)や割り込みコントローラやDMAコントローラ，シリアルコミュニケーションインターフェースなど，CPUのペリフェラルデバイスや，一つのCPUを疑似的に並行処理させることを可能にするリアルタイムOSなどです．

生産性向上を支える助っ人達は，過去の障害や現在の開発における不具合の現象や原因，対策をデータベース化した不具合データベースや，ソフトウェアの資産を構成管理しフィールドで問題が起こったときのトレーサビリティを迅速にするための構成管理ツール，ソフトウェアのソースコードを静的にテストし未然にバグを防ぐための静的テストツールなどがあります．

これらの助っ人達を特徴分類の重みによって，上手に使い分けることが，開発コストと商品価値の良いバランスを保つためのポイントです．要求がそれほど高くない部分に高価な助っ人を導入することは，組み込みシステム全体の利益には決してなりません．ただし，組み込み機器が商品群としてラインナップをそろえている場合は，助っ人が高価でも商品群としてはリーズナブルな場合もあります．たとえば，要求の多様化で登場した32ビットマイコンをローエンド機器で使用するのは無駄かもしれませんが，生産性向上を支える助っ人達で登場した不具合データベースや構成管理ツール，静的テストツールをローエンドの機器に使うことは商品群全体としてペイできるのであれば，決して不利益ではありません．

5 組み込みソフトウェアの特徴の分類に対する配分は，今後どうなっていくのか？

組み込み機器にとって静的制約は，常に変化するものです．CPUの消費電力，パフォーマンス，ROM，RAMの容量はどんどん向上していきます．また，PLDの低価格化，低消費電力化が進めば，組み込み機器の心臓部としてCPUとさまざまな周辺デバイスを組み合わせた1チップマイコンを選択するのではなく，CPUはコアだけを使用し，周辺デバイスはFPGAの中でIP(Intellectual Property：知的財産)を組み合わせることによって実現するという選択肢も出てくるでしょう．CPU自体がFPGAの一つのモジュールになる可能性もあります．

したがって，静的制約は時の流れ(技術の発展)につれて，その制約が小さくなる方向に向かいます．一つのCPUが十分に普及すれば，そのCPUシリーズの性能は向上しても価格は据え置きか，かえって安くなることさえあるでしょう．静的制約は，ハードウェアのテクノロジの進歩でどんどん緩和されていく要件です．

一方，組み込みの組み込みたるゆえんは動的特性，すなわちリアルタイム性による部分が大きく，この部分は時間の流れとは関係なく普遍であると考えられます(いずれ，Windowsのようなリアルタイムでない OSでも，リアルタイム性が実現できてしまうほど静的制約が小さくなる日もあるかもしれないが，それはまだまだ当分先の話だろう)．

そう考えると，ハードウェアの発達で緩和された静的制約の部分を，要求の多様化や生産性向上のほうに回していくというのが，これからの組み込みソフトの流れではないでしょうか．ただし，組み込み機器の場合，構成部品のコストを下げるという命題がなくなることはないので，パフォーマンスの高いCPUや資源を使うことでコストアップすることと，再利用性や保守性が上がることを天秤にかけ，いちばん釣り合いの良い点を見つけることが重要です．

また，パフォーマンスの高いCPUはハードウェアエンジニアリングによって作り出されますが，再利用性や保守性を上げ

るのはソフトウェアエンジニアリング(要するに人間の頭)によって作り出されるので，ソフトウェアの作成が完全自動化されないかぎり，組み込みエンジニアへの教育は，組み込み機器の開発にとって必要不可欠です．

6 組み込みシステムの特徴における比重の変化と商品開発の考え方

開発しようとしている商品が，組み込みシステムの特徴分類(**図2**)において静的制約の比重がとても大きい場合，リアルタイムOSを載せる余裕さえない場合もあるでしょう．しかし，商品の小ささや価格の安さがユーザーの望んでいるもっとも重要な要素なのであれば，それも仕方ありません．リアルタイムOSを使わなかったことによる再利用性，保守性，信頼性の低下が，後々リコールなどのユーザーの不利益につながらなければ，開発効率よりユーザー要求を優先させるのは当然です．たとえば，目標価格2,580円の強中弱の3段階しかないハンディ肩たたき機にリアルタイムOSを搭載して再利用性や保守性が良くなったとしても，OSのライセンス料が材料費として1台あたり50円上乗せされれば，ユーザーにとっては100円以上の負担となり，信頼性がアップしたという利益より，価格が上がったという不利益のほうが大きく感じられるのです(**図3**)注2．

Column
組み込みソフトウェアとプログラムの改変について

組み込みソフトウェアの特徴の一つに，プログラムがROMに搭載され，後からプログラムを書き換えることができないという点があります．ROMに書き込まれる組み込みソフトウェアのプログラムは製品出荷時に固定されて，後でROM自体を書き換えることができないので，商品のリリース後に不具合が発生したり仕様を変更したくなったりしても，気軽にプログラムを交換するわけにはいきません．

一方パソコンでは，BIOSを除けば，使用するプログラムをユーザーが自由に選択できます．このため，プログラムはCD-ROMやフロッピディスクなどの携帯用記憶媒体に保存された状態で販売されるか，インターネットなどを通じてダウンロードファイルという形式で供給されます．ユーザーの選択の自由度は高く，基本ソフトと呼ばれるOSでさえこのような形式で供給されます．パソコンの世界では，ハードウェアとソフトウェアが十分に分離されていると考えてよいでしょう(「コンピュータ，ソフトなければただの箱」というフレーズが流行った時期があった)．

パソコンの場合，ハードウェアとソフトウェアの分離が進んでいるため，異なるハードウェアでもOSが同一であればアプリケーションソフトは別々のプラットホーム上で動作できるというメリットがあります．しかし，ハードウェアがブラックボックスのままで，ユーザーが要求する仕様の性能の程度を特定することはできません．パソコンのアプリケーションソフトウェアには動作させるパソコンの推奨性能などがパッケージに記載されていますが，すべてのパソコンでアプリケーションソフトウェアの動作の最適性を保証することは不可能です．

逆に，気軽にプログラムを置き換えることができない組み込みソフトウェアは，ハードウェアとの関係がより深くなります．汎用目的のパソコンとは異なり，特定の機能を提供する専用機ですから，ソフトウェアとハードウェアは相互に協力し合いながら，最高の性能と品質および価格を追求することが求められます．このため，ハードウェアの変更に対してソフトウェアの修正を最小にするといった生産性向上を目的としたハードウェアの仮想化やプログラムの階層化は，性能や価格を犠牲にするという理由からある程度あきらめざるを得ませんでした．組み込みソフトウェアの世界では，抽象化という概念は必ずしも絶対的善とはされてこなかったのです．しかし，組み込み機器にもパソコン並みの多様な用途が要求されたり，システムが大きくなりソフトウェアの安全性・信頼性を商品発売時に固定できなかったりするようなものも出てきています．このようなハイエンドな組み込み機器では，ハードウェアの仮想化やプログラムの階層化も必要になります．現にレーザープリンタなどのソースコードが100万行を超えるような多用途の組み込みソフトウェアシステムでは，これまでの小規模な組み込み機器には見られなかったようなソフトウェアの抽象化が不可欠になってきています．

ところで，組み込みソフトウェアがROMで提供されることが多いのは，用途が限定されているのでプログラムを変える必要がないという点と，材料費をおさえる必要性からです．しかし最近は，ユーザーからの要求の多様化や開発期間の短縮にともない，フラッシュメモリなどのプログラムを書き換え可能な媒体で供給する製品が増えてきました．これらの製品は，機能追加や性能向上あるいはバグの修正などの理由で，商品搭載プログラムを出荷後に更新できる機能を備えています．このようにプログラムの書き換えが可能な商品では，更新用のプログラムをWebページで公開している企業も数多くあります．

しかし，ユーザーにとってこれは本当に喜ばしいことなのでしょうか？　購入する時点で最高のものを提供してほしいとユーザーが要求するのは当然であり，また，アップグレードの煩わしい作業をユーザーに要求するのはメーカーの責務をユーザーに押しつけることにもなりかねません．安易なバージョンアップ路線に走らないように，また割高なコストをユーザーに転嫁することがないように，組み込みソフトウェアの開発者は肝に銘じる必要があります．

注2：OSのライセンス料は，無制限のライセンス契約をしたりオープンソースのOSを利用したりすれば，材料費には含まれないと考えることができる．

しかし，さまざまなもみほぐしパターンのある目標価格13,000円のハンディ肩たたき機ではどうでしょうか？　複雑なパターンのモータ制御や機器の一次故障時の素早い安全装置の起動など，装置全体における動的特性の比重が高いので，リアルタイムOSは搭載したほうがよさそうです．OSのライセンス料50円も，材料費3,000円の中で何とかなります．

また，肩たたき機のラインナップがたとえば7機種もあり，肩たたき機商品群が構成されている場合はどうでしょうか？　ローエンドの肩たたき機1機種を考えた場合，リアルタイムOSの搭載は必要なかったかもしれませんが，ローエンドからハイエンドまでの肩たたき機商品群を考えて，リアルタイムOSをすべての機種に搭載することはできないでしょうか？　肩たたき機の機構制御部分のソフトウェアコア資産を商品群の共通資産とすれば，CPUもローエンド機からハイエンド機まですべての商品で共通とすれば，CPUの使用量がローエンド機で年間1万個だったものが，商品群全体で年間10万個になり，1個あたりの材料費がたとえば500円から300円に下がります．そうすれば，リアルタイムOSのライセンス料50円が材料費に上乗せされてもコストダウンは達成できます(**表1**，**図2**，**図4**)．

7 組み込み機器と組み込みソフトウェアへのアプローチの仕方

これまで分析してきたように，組み込み機器と組み込みソフトウェアはいろいろな側面からの特徴をもっており，それぞれの特徴に対する比重も，商品群の存在や新しいCPU，周辺デバイスの出現によって一つのグレードにとどまることなく常に変化していくことが，おわかりいただけたと思います．

組み込み機器に対する要求分類のグレードが変化するということは，組み込み機器に搭載されるソフトウェアに期待される機能や性能も当然変化していくことになります．また，ソフトウェアに対する要求が変わるのであれば，ソフトウェアを開発するための手法も変化させていく必要があるかもしれません．重要なことは，組み込み機器や組み込みソフトウェアへの要求は時が経過するにつれ必ず変化するものであるという認識を開発者がもつことです．

組み込み機器では，ハードウェアの変更には大きなコストが発生するために，変化に対して柔軟性をもつソフトウェアで市場要求の変化に対応しなければいけないことがよくあります．このため，市場要求の変化に追随する使命を帯びた組み込みソフトウェア開発を行う際には，変わりやすいものへの柔軟な姿勢を常に維持していることがたいせつです．

参考文献

1) (株)ルネサステクノロジ，「半導体専業メーカーの強みを発揮し最終セットの競争力を支援する幅広いソリューションを提供」，『RENESAS EDGE』，創刊号

さかい・よしお　組込みソフトウェア管理者・技術者育成研究会(SESSAME)
いなば・みちお　ブラザー工業(株)

Appendix
カラーで見るUML図（ユースケース図，クラス図，コラボレーション図）

酒井由夫

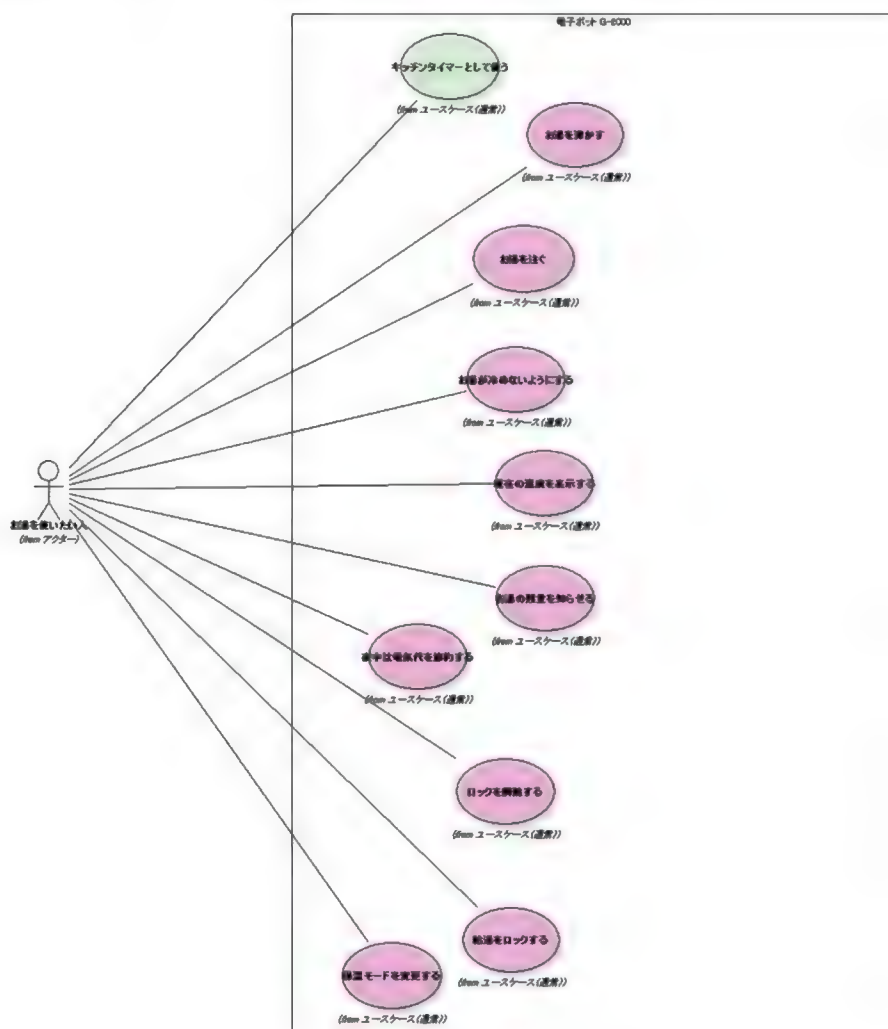
本特集の第3章（ドメイン分析）ではUMLのパッケージ図について，第4章（設計・実装）では電子ポット商品群のコア資産について，UMLのユースケース図，クラス図，コラボレーション図を書いて設計を進めています．

記事の中に登場する普及型電子ポット「G-2000」と近未来型電子ポット「G-7000」のUML各ダイアグラムは，ダイアグラムの要素を色分けし，色に一貫性をもたせることで，コア資産の種類を識別し視覚的な効果を高めるようにしています．そこで，これらのUML図をまとめて，ここで一覧的に紹介します（**表1**，**図1**～**図14**）[注1]．各図のくわしい解説は，以降の章で行います．

さかい・よしお　組込みソフトウェア管理者・技術者育成研究会（SESSAME）

〔**表1**〕**一覧表**

図	内容
図1	G-2000ユースケース図
図2	G-7000ユースケース図
図3	G-2000詳細ユースケース図
図4	G-7000詳細ユースケース図
図5	G-2000保守ユースケース図
図6	G-2000ドメイン構造図
図7	G-7000ドメイン構造図
図8	G-2000ドメインマッピング（保守）
図9	共通コア資産「お湯を沸かす」クラス図
図10	共通コア資産「お湯を沸かす」コラボレーション図
図11	共通コア資産「お湯を注ぐ」クラス図
図12	共通コア資産「お湯を注ぐ」コラボレーション図
図13	共通コア資産「節電する」クラス図
図14	共通コア資産「節電する」コラボレーション図

〔**図1**〕**G-2000ユースケース図**

注1：UMLの各図は，本誌2003年8月号「日本語が使えるUMLツール最新比較」の記事でも紹介したツール「Enterprise Architect」（スパークスシステムズジャパン）を使って作成している．本特集記事で使用したEnterprise Architectのプロジェクトファイルおよび Enterprise Architect本体の30日限定特別評価版をInterface誌のホームページ（`http://www.cqpub.co.jp/interface/`）からダウンロードできるようにする予定．

〔図2〕G-7000 ユースケース図

〔図3〕G-2000 詳細ユースケース図

〔図4〕G-7000詳細ユースケース図

〔図5〕G-2000保守ユースケース図

〔図6〕G-2000ドメイン構造図

〔図7〕G-7000ドメイン構造図

New Products ―パソコン周辺機器の電源向けPWMコントローラ「IRU3039」
インターナショナル レクティファイアー ジャパン(株)は，PWMコントローラ「IRU3039」を発売した．DDR SDRAMやグラフィックカードなどの電源向けの製品で，スイッチング周波数は200k～400k，最小出力電圧0.8V，スイッチング素子は外付けである．

〔図8〕G-2000ドメインマッピング(保守)

〔図10〕共通コア資産「お湯を沸かす」コラボレーション図

〔図11〕共通コア資産「お湯を注ぐ」クラス図

〔図9〕共通コア資産「お湯を沸かす」クラス図

〔図12〕共通コア資産「お湯を注ぐ」コラボレーション図

〔図13〕共通コア資産「節電する」クラス図

〔図14〕共通コア資産「節電する」コラボレーション図

News Flash ―― 英ARM社，STARC/ASPLAと提携し90nmプロセスでのARMコアの検証を開始
英ARM社は2003年9月4日，(株)半導体理工学研究センター(STARC)および(株)先端SoC基盤技術開発(ASPLA)と提携し，同社のARMコアをASPLAの90nmプロセス上で検証すると発表した．検証作業は9月から3社協同で行われる．

第2章 電子ポット商品群にプロダクトラインを適用してみる

体系的な再利用で実現する商品開発

酒井由夫/今関 剛

第１章で考察した，組み込み機器の特徴である「変わりやすい部分」と「変わりにくい部分」に着目し，「プロダクトライン」という考え方を導入して，開発を効率化する手法について解説する．また，「変わりにくい部分」を体系的に再利用する方法も説明する．ここでは「電子ポット商品群」を例として取り上げながら解説する．

（編集部）

1 組み込み機器・組み込みソフトウェアの「商品企画」

組み込み機器の場合，顧客に満足してもらえる商品をリリースできるかどうかは，ユーザー要求をハード・ソフトの技術でどれだけ実現できるかどうかにかかっています．ビジネス系のアプリケーションソフトウェアのように，パソコンやワークステーションなどのプラットホームが定まっていて，ソフトウェアだけで要求を実現するのとは異なり，**組み込み機器の場合はハード・ソフトのバランスが要求実現のカギ**になります．

どのようなバランスになるかは，第１章で述べたように，対象となる商品または商品群によってまちまちですが，どんな組み込み機器でも，ユーザー要求をどこまで取り入れるかの判断をハードウェアでの機能実現の可能性とソフトウェアでの機能実現の可能性を十分に理解しているエンジニアが行えば，良い商品を作り上げることができます．ハード・ソフトの制限をよく知っているエンジニアがそれらの制限を意識しすぎて発想の転換ができない場合もありますが，多くの場合は実現の可能性をポジティブに考えることで，最良の選択を行えるはずです．

コミュニケーションギャップを起こさないためには，ハード・ソフト両方の知識を持ち合わせた一人のエキスパートが，商品企画について実現の可能性を検討するのが最適です．しかし，現実的にはハード・ソフトの役割は分担されており，ハードウェアのスペシャリスト，ソフトウェアのスペシャリストがそれぞれ相手の領域にどれくらい踏み込めているか，また，ハードウェアエンジニア，ソフトウェアエンジニアのコミュニケーションがどれくらい密に行われているかが，顧客の要求を実現できるかどうかのバロメータになります．

とくに，機器のダウンサイジング，低消費電力化，コストダウンを実現するには，ハード・ソフトのトレードオフが不可欠です．なぜなら，ハードウェアを削減して，ソフトウェアに負担をかけるとCPUパフォーマンスが足りなくなり，結果的に顧客の要求を削らざるを得なくなるかもしれません．一方，ソフトウェアの負担を軽くするためにハードウェアにどんどん機能を移譲していけば，基板面積が広くなり，消費電力も余計にかかり，コストもアップしてしまいます．

その微妙なバランスを判断できるのが，顧客の要求を熟知したハード・ソフトのエキスパートであり，エキスパートチームであるといえます．チームでこの判断を行う場合は，ディスカッションするメンバが腹を割って話し合える間柄でなければなりません．なぜなら，チームが企画担当者とエンジニアから成る場合，顧客の要求を熟知した企画担当者が「この機能はぜひ実現したい」と思っても，ハード・ソフトのエンジニアに本当はできるかもしれないのに「それは技術的に不可能です」と言われてしまえば，話がそれで終わってしまうからです．「技術的に不可能です」という言葉の裏には「検討するのがめんどうだ」とか，「やりたくないという気持ち」が隠れていて，そう言わせているのかもしれないのです．

企画担当者が熱意をもってエンジニアを説得することで立場を逆転させられるかもしれませんが，エンジニアが顧客の要求を十分に理解し，機能を実現したときの顧客満足度がどれくらい高くなるのかをきちんと判断できれば，技術的な難易度と顧客満足度の最適なトレードオフを行うことができ，妥協のない良い商品を生み出すことができるでしょう．

2 組み込み機器の要求仕様について

組み込み機器の仕様が，商品企画段階で完全にフィックスしていることはまれです．組み込みソフトウェアは，**変わりやすい部分と変わりにくい普遍的な部分の両方の側面**をもちあわせています．これは，組み込みソフトウェアが毎回ゼロから開発するのではなく，たいていの場合，現行機種からの機能追加や機能アップといった差分開発が中心であるからです．そのような観点から考えると組み込みソフトウェアでは，おおざっぱにいえば組み込み機器を商品群としてとらえれば，商品群としての基本機能を実現するソフトウェアモジュールがコア資産であり普遍的な部分で，ユーザーインターフェースの部分が変わり

やすい部分であるといえます．

したがって，組み込みソフトウェアの場合商品群としてのコア資産のソフトウェアを要素技術としてじっくり開発し，ユーザーインターフェース部分についてXP（eXtreme Programming [1]）的な繰り返し開発を行うという「合わせ技」を使うことで，効率の良い商品開発を行うことができると考えられます．

3 商品群としての組み込み機器開発

組み込み機器のソフトウェア開発と，業務系のソフトウェア開発の違いは何でしょうか？　業務系のソフトウェア開発は，どちらかといえば「単発的な開発」であるのに対して，組み込み機器のソフトウェア開発はある特定の市場に対して投入する商品群の開発であることがほとんどで，それらの商品群の中には共通のコアとなる資産が存在し，その共通のコア資産をベースにバリエーションをもたせて新しい商品を作るという開発スタイルを取ることが多いと考えられます．

このような単発的な開発と，コア資産をベースにしてバリエーションで新たな商品を作り出す差分開発は，取るべき開発のスタイル，プロセス，戦略はおのずと異なります．

もちろん業務系のソフトウェアも商品ラインナップをもち，差分開発によって新しいバージョンの商品が作り出されることはありますが，組み込み機器の商品群ほどローエンドからハイエンドまでといった幅広い商品グレードで，またソフトウェアのコア資産はもちろん，人材や商品開発のプロセスが再利用されることは少ないと考えられます．したがって，組み込み機器・組み込みソフトウェアを効率良く，短期間で開発するためには，コア資産の再利用率が最大になるような戦略を取る必要があります．コア資産の再利用率を最大にするということは，言葉でいうのは簡単ですが，実際にはさまざまな障壁を乗り越えなければ実現できません．

たとえば，組織が縦割りになっており独立採算性を採用しているため，隣の部署のソフトウェア資産を簡単に利用できないようなしくみになっているとか，エンジニア同士のコミュニケーションが悪いため意図せずに同じような機能をもつソフトウェア資産を作っていたとか，他の部門で作った資産があるにも関わらずそのソフトウェア資産の信頼性が把握できないため，一から作り直してしまったなど，再利用の促進を阻害する障壁はたくさんあります．

〔図1〕C＊Oの位置付け

縦割り組織に横軸を通すための組織機構として，「C＊O」と呼ばれる，いろいろなスコープで組織全体を統括する責任者が設置されることがあります（**図1**）．「C＊O」はChief＊Officer（＊統括責任者）のことで，CEO（Chief Executive Officer：最高統括責任者）はよく聞かれる言葉になりましたが，CIO（Chief Information Officer：情報統括責任者）やCTO（Chief Technology Officer：技術統括責任者）など，全社的なインフラストラクチャや全社的な技術の統括を行うために，縦割り組織のセクショナリズムにしばられない統括責任者も，多くの企業で設置されるようになりました．

しかし，このような縦割り組織に横軸を通すための統括責任者が存在しなかったり，CTO（技術統括責任者）がソフトウェアのコア資産を積極的に共有化する取り組みに理解を示してくれなかったり，資産の共有化を促進する部署がなかったら，どうやってコア資産の共有化を進めていけばよいのでしょうか．

開発プロセスや人材の育成方法の再利用については，会社の外側からの規制によって促進できる場合があります．たとえばISO9001では，開発のプロセスを明確化し，プロセスが想定したように運用されていることを計測する手段をもち，計測した結果をフィードバックしてプロセスの改善を行うような品質マネージメントシステムが組織に要求されていますし，CMM（ソフトウェア能力成熟度モデル）では組織としての成熟度が客観的に計測され，レベルが明確化されます．ISO9001の取得を宣言することで，品質マネージメントシステムを構築するということは，開発プロセスや人材育成の方法，リスクマネージメントの方法を明確化し再利用することにほかなりません．また，CMMのレベル3，4，5をめざすような成熟した企業は，組織間の障壁を越えて資産の再利用を推進できるはずです．

このようなISO9001やCMMを取得することをめざしている企業であれば，開発プロセスや教育，リスクマネージメントについての再利用活動へのアプローチへ導くことは，それほど難しくはないでしょう．

しかし，商品価値の向上に直結するいちばん重要なソフトウェア・コア資産の再利用は，このような外側からの要因では進みにくいものです．なぜなら，ソフトウェア・コア資産はハードウェアのコア資産と違って，商品開発のプロセスの中で現実的な形として見えにくく，また，これを再利用したことによって直接的にどれだけコストが削減できたか，または開発期間が短縮されたのかが計算しにくいという特性をもっているからです．

したがって，縦割り組織の垣根を越えて，ソフトウェア・コア資産の再利用率を最大限に上げ，ソフトウェアエンジニア全

体がその効果を実感し，顧客に再利用によって向上した品質や総合的なコスト，デリバリ期間の短縮を感じてもらうには，再利用についての考え方や方法論をアカデミックに学び，正しく適用する必要があります．再利用についての理論をきちんと理解し，実践することで，さまざまな障壁を乗り越えることができるのです．

本稿では，この再利用の科学である**プロダクトライン**を実践的に利用した具体例を取り上げますが，自分たちの業務ドメインにぴったりとした利用を試みたいのであれば，**プロダクトライン**自体の考え方をきちんと学習することをおすすめします．プロダクトラインを学ぶには，本家本元のカーネギーメロン大学ソフトウェア工学研究所のWebページを読むことで可能です．日本において実例をもとに導入を検討する場合は，組み込みソフトウェアエンジニアのコミュニティであるEEBOF（組込みソフトウェア管理者・技術者育成研究会：Working Group1 URL：http://www.bof.jp/eebof/）に参加するのが近道の一つでしょう．

4 プロダクトライン入門

商品群のコア資産を共有し，再利用して新しい商品を開発する戦略は**プロダクトライン**と呼ばれ，ソフトウェア開発における**プロダクトライン**の適用については，前述のとおりカーネギーメロン大学のソフトウェア工学研究所で研究が進められています．

プロダクトラインは，組み込み機器開発だけに適用する開発手法ではないですが，商品や商品群が求められる市場性やこれまでの組み込み機器の開発スタイルを考えると，**プロダクトライン**の考え方は組み込みソフトウェア開発にフィットすると考えられます．

プロダクトライン開発の考え方を組み込み機器・組み込みソフトウェアに適用し，市場ニーズにマッチした商品を効率よく開発するには，まず，自分たちの業務ドメインを分析し，かつ，すでに市場に投入されている商品の特徴から商品のコア資産の固まりを摘出・分類することが重要です．たいていの場合，このようなコア資産の摘出・分類は暗黙のうちに行われていることが多いのですが，コア資産の再利用・共有をきちんと意識して体系的に行い，組織全体としての活動へ広げることが，**プロダクトライン**戦略の目標です．

● **ソフトウェアプロダクトラインの定義**

カーネギーメロン大学ソフトウェア工学研究所において，「**ソフトウェアプロダクトライン**とは何か」が，次のように紹介されています．

A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.

この文章を意訳すると，次のようになります．

ソフトウェアプロダクトラインとは，次のような特徴をもったソフトウェアの集合体である．

1）そのソフトウェア集合体は，特定の市場または特定の用途において明確なニーズを満たすように管理されている．

2）そのソフトウェア集合体は，商品群のコア資産から定められた方法によって作り出される．

プロダクトラインに関しては，最近参考文献5）が発刊されたので，興味のある方は読んでみてください．

5 体系的な再利用と再利用の対象の推移

ソフトウェアの再利用には，場当たり的な再利用と体系的な再利用の2種類があります．場当たり的な再利用とは，オブジェクトやコンポーネント，アルゴリズムのライブラリなどを開発者が個人的に再利用することで，体系的な再利用とは個人単位ではなく組織単位で行う，目的をもった再利用のことです（**表1**）．

〔表1〕場当たり的再利用と体系的再利用の違い

場当たり的再利用	体系的再利用
個人単位で行う再利用	組織単位で行う再利用
シーズ志向の再利用	市場動向，技術動向を考慮した顧客志向の再利用
再利用の対象はソフトウェアのみ	再利用の対象はソフトウェアのみならず，人材や開発プロセスにも及ぶ
目的の明確でない再利用	目的が明確な再利用
再利用活動がマネージメントされない	再利用活動をマネージメントする

表1を見ていただければわかるように，場当たり的再利用は再利用活動がマネージメントされていないため，必ずしも効率が良いとは限りません．それに対し，体系的再利用は明確な目的のある再利用であり，再利用活動がマネージメントされているので効率の良い再利用になります．再利用が「効率的であるようにマネージメントされている」ともいえます．

また，場当たり的な再利用と体系的な再利用のもっとも重要な違いは，場当たり的再利用がエンジニア自身の都合が重視されたシーズ志向の再利用であるのに対し，体系的再利用は市場動向や技術動向を考慮した顧客のためのニーズ志向の再利用である点です．体系的再利用が目的のある再利用になるのは，この再利用活動を顧客の利益に明確に結びつけているからです．再利用活動は縦割り組織の障壁を乗り越えて行わなければならないので，最終的には顧客の利益に結びつくことを理論的に説明しながら行っていかなければ，いつの間にか活動に理解が得られないか，活動推進者のみに負担が強いられることになってしまいます．このような状況に陥らないためにも，再利用を推進できるような組織構造を作り，計画を立てて実行し，リスク

を管理することが重要です．

6 プロダクトラインの三つの活動とは？

プロダクトラインでは，次の三つの活動によって体系的な再利用を実現していきます(**図2**)．

1）ドメインエンジニアリング→(キーワード：摘出)

2）アプリケーションエンジニアリング→(キーワード：利用)

3）マネージメント→(キーワード：フィードバック)

1）ドメインエンジニアリング

ドメインエンジニアリングとは，商品群の共通する機能や特性を摘出し，それらをまとめて再利用できる形式にする活動のことです．ドメインエンジニアリングで重要なのは，市場や技術の動向，顧客の要求仕様の変化を予測し，変化に追随してバリエーションをもたせられるようにコア資産を構築することです．この活動には，すでにある商品やこれから開発する商品の具体的な仕様や実装のイメージから，共通する機能や特徴を摘出し抽象化しなくてはならないため，高い分析力と抽象化能力が必要です．この分析と抽象化をより容易に行えるようにするために，ロードマップやドメイン構造図といった，***プロダクトライン***の道具が用意されています．***プロダクトライン***を実施するためのこのような道具を使えば，コア資産を摘出し再利用へ導くことが容易になります．

ドメインエンジニアリングによって摘出されたコア資産は，ソフトウェアの資産だけとは限りません．次のようなものもコア資産となります．

- 要求仕様を分析した要求モデルとドメインモデル(**コラム**参照)
- ソフトウェアのアーキテクチャ(実装方法)
- フレームワーク(構造・枠組み)とコンポーネント(ソフトウェアのモジュール)
- ノウハウやパターン(商品群に特化した注意点や定石)
- テスト計画やテストケース(テストの方法やテストケースの作り方)
- クックブック(コア資産の使い方を示した仕様書)
- 開発・管理プロセス(開発や管理の流れ)
- 教育やコンサルティングの方法(教育およびコンサルテーションの目標や方法)

コア資産を提供する側のエンジニアが考慮すべき点は，コア資産を利用する側の立場に立ってインターフェースを設計することです．コア資産の提供者は，コア資産の利用者に対して必要なインターフェースのみを公開し，利用者に公開する必要のないコア資産内部の機能に起因する部分は隠ぺいするように設計を行うべきです．また，コア資産を利用するユーザーは一人ではないので，利用者の違いを吸収できるようなしくみもコア資産に仕込んでおく必要があります．

さらに，コア資産の提供者は利用者に対し，どのようにしてコア資産を使えばよいのかの利用手引き書を作る必要があります．このような利用手引き書は「クックブック」と呼ばれることがあります．クックブックは，料理の作り方が書かれたレシピです．料理の初心者でも簡単に目的の料理が作れるかどうかは，クックブックのできしだいということになります．

クックブックには，次のような項目を記述します．

- コア資産の機能・性能(コア資産を利用する目的)
- コア資産の利用にあたっての制約条件
- コア資産の機能・性能が実装できたことを確認するための判断基準
- コア資産の機能・性能を確認するための手順
- コア資産の機能・性能を確認するためのツール

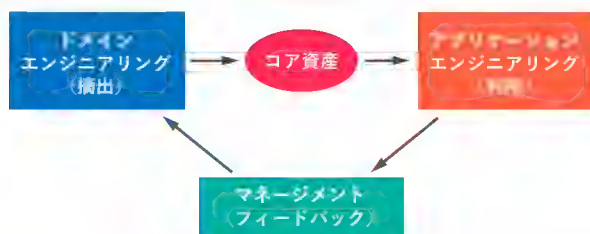
これらの項目を見ていただければわかるように，コア資産の提供者には，高度な抽象化能力とコア資産の利用者をエンドユーザーと見立てた細かい気配りが必要になります．コア資産の提供者になれるエンジニアはそれほど多くないので，組織はこのような能力のあるエンジニアをピックアップし，専門の教育を行う必要があります．

2）アプリケーションエンジニアリング

アプリケーションエンジニアリングとは，ドメインエンジニアリングによって作り出されたコア資産をもとに，派生したモジュール(アプリケーション)を作る活動のことです．つまり，アプリケーションエンジニアリングとは，料理の例でいうと材料とクックブックから料理を作ることになります．

アプリケーションエンジニアリングでは，コア資産の提供者に要求された抽象化能力とは逆に，抽象化されたコア資産を具象化し実際に商品に実装する能力が要求されます．アプリケーションエンジニアリングでは，ドメインエンジニアリングで構築された共通資産を再利用して，バリエーション商品を作り出す差分開発を行うことになります．しかし，そのバリエーションとは，刻々と変化する顧客の要求に対応するためのユーザーインターフェースに起因することが多いため，ユーザーニーズを的確に把握し，実現するための能力が問われます．

しかし，従来の商品開発のようにコア資産の開発とアプリケーションの開発を同じエンジニアが同じようなフェーズで行わなくてもよいので，アプリケーションエンジニアはコア資産の実装についてはそれほど工数を割く必要はなく，最適なユーザーインターフェースの構築に専念することができます．

〔図2〕プロダクトラインの三つの活動

3）マネージメント

最初に作ったコア資産やクックブックが最適で完全であることはあり得ません．どんなに優秀なドメインエンジニアが作ったコア資産やクックブックでも，それらを継続的に管理し保守する活動は必要です．アプリケーションエンジニアリングにおいて，コア資産やクックブックが適切に利用されていることを指導・監視することも大事ですが，コア資産やクックブックに不都合な点や追加すべき機能が出てきたときは，アプリケーションエンジニアがドメインエンジニアに対し，その内容をフィードバックする必要があります．これを受けてドメインエンジニアは，コア資産やクックブックをより良いものに洗練することができるのです．

7 ロードマップとドメイン構造図について

1）ロードマップ

ドメインエンジニアリングにおいて商品群の特徴を摘出する

Column

狭義のドメインと，広義のドメイン

一般的なソフトウェア開発では，スコープを決めて開発を進めようとしますが，ドメインを決めてから開発を進めることはきわめてまれです．ここでいうスコープとは，ソフトウェアに求められる要求のセットであり，ドメインは，これらの要求の組み合わせを考えることができる一定範囲の領域を意味します．具体的には，似たようなソフトウェアが複数存在しているとき（たとえばワープロソフトと簡易文書エディタなど），それらソフトウェア群は単一のドメイン（たとえば文書編集ソフトウェアドメイン）に存在するととらえることができ，個々のソフトウェア機能は，それぞれ重複があるものの，別々の要求セットを満たしているということができます．

通常，業務系のソフトウェア開発では受託/請負の形式をとります．また，成果物納品後はシステムの運用として機能拡張および保守に工数を割くこととなります．このようなビジネスモデルを前提とした開発スタイルでは，ソフトウェア群の開発や複数の開発プロジェクト，ソフトウェアにまたがる共通性を抽出する機会や動機が少なく，Scratch&Build型開発を前提とした，適度に抽象化されたフレームワークが安価で提供されている場合もあり，ドメインを決めて（あるいは分析して）から開発を進められていないのが実情です．

「ドメインを決める」とは，ある分野の熟練者達（あるいは顧客そのもの）が業務上理解している一連の概念と用語で特徴づけられた知識または作業の領域を定義するものです．一般的なソフトウェアの開発では，顧客の要求内容やそこで使用されている専門用語（業界用語）を開発担当者が正確に把握するため，専門用語をかみくだいて用語辞書に定義する必要があります．これにより，顧客および熟練者と開発担当者の誤解を未然に防ぐことができ，用語に対する共通の認識を保持することが可能となります．逆に，このような用語辞書の作成は，ドメインが分析できていることを保証するための唯一の手段となっています．

これら用語辞書に定義された用語の中には，あるソフトウェア群に共通して取り扱われる情報や処理の内容が含まれており，これをもとにフレームワークを構成することにより，特定のドメインで再利用できる大規模なソフトウェア部品を実現することが可能となります．その結果，以降のシステムの開発コスト削減に大きく貢献することとなります．このような，あるソフトウェア群を認識することができる範囲をドメインととらえる見方をここでは「狭義のドメイン」と呼びます．

では，組み込みソフトウェア開発を，同じ視点からながめてみましょう．組み込みソフトウェア開発組織では，新たに企画された製品に対するソフトウェア開発であっても，ソースコードの流用をベースとした再利用型開発が一般的であり，何年も前のソフトウェア資産を活用しているのが実情です．これは，開発するソフトウェアのドメインに変化がなく，新規に開発し直す必要がないことに起因します．しかし，従来の資産を流用できているからといって，ソフトウェア開発プロジェクトが成功しているかというと疑問です．過度な流用により，拡張しにくい既存ソフトウェアを改造して対応しなければならない状況であり，担当者の負荷が改善されにくい悪循環に陥っているのです．

この悪循環を断ち切る手段はいろいろありますが，このような悪循環を排除できたと仮定しましょう．製品開発組織として複数のソフトウェア群を効率よく生産するためには，提供していく商品のロードマップをベースに，製品毎のソフトウェア・バリエーション，共通部分，および開発活動における繰り返し作業などをあらかじめ抽出し，投資対効果を見据え，中長期的にソフトウェア資産を再利用していく枠組み（***プロダクトライン***）を考える必要があります．このような，開発スタイルを含めた再利用を前提とした開発を行うためには，商品ロードマップやソフトウェア開発技術，既存ソフトウェア資産，組織体制を，事前に分析しておく必要があります．これらを「広義のドメイン」分析と呼びます（***プロダクトライン***分析は広義のドメイン分析の一種と呼べるであろう）．広義のドメイン分析の成果物の形式は，狭義のドメイン分析のように明確な形式ではなく，さまざまな形式を考えることが可能です．

少々難しい話になりますが，そもそもドメイン分析は，「問題空間」と「解空間」との対応関係を抽出することに価値があります．「問題空間」に存在する頻出する開発上の課題や実現すべき要求をグルーピングし，それに対する有効なソリューションを「解空間」から選択することが重要です．従来のCASEツールや方法論，プロセスなどは「解空間」の一部でありながら，「問題空間」を絞り込まない（定義しない）まま，薬でいうところの特効薬的な形で広まりました．この弊害として，ツールを導入したけれど効果が現れなかったり，技術の誤用・濫用を行ってしまったりする開発組織が多く存在しています．

には，ロードマップを描くことが有効です．ロードマップは，対象となる商品群に対して横軸を時間，縦軸を空間に取って描きます．対象とするロードマップは，商品自体のロードマップはもちろんのこと，商品群に関係する規制のロードマップ，規制ロードマップから展開した技術ロードマップ，また，市場動向や技術動向を予測して描いたロードマップ，商品戦略に基づいた市場誘導のロードマップなどがあります．

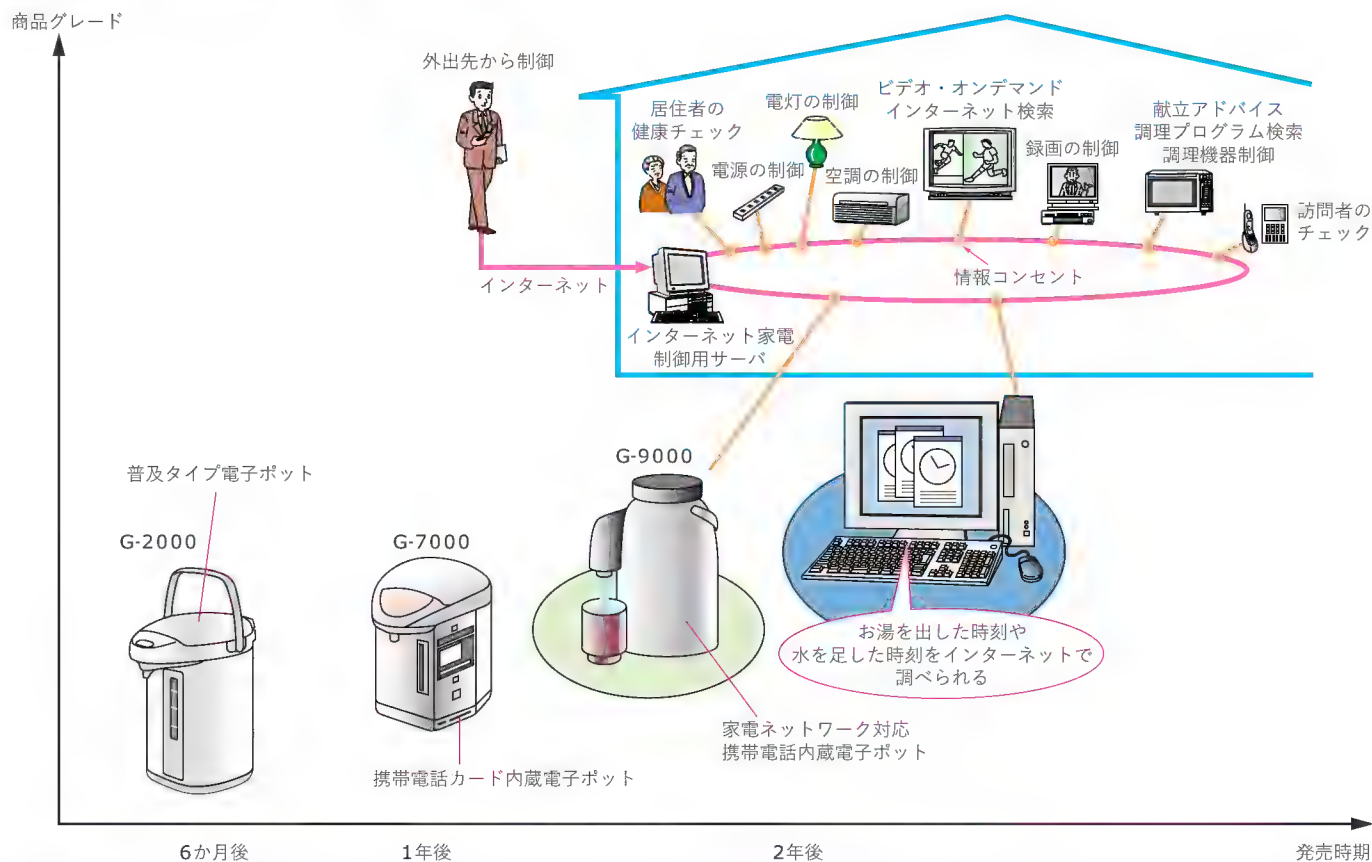
このようなロードマップを描くことによって，現存する商品群と未来の商品群を合わせた全体の商品群に共通するコア資産を具体的に思い描けます．もちろん，予測した未来が実際と異なることはありますが，ロードマップを作った上で戦略的に再利用活動を行っていけば，未来の予測に対する確度も徐々に上がっていくはずです．

ここからは，より具体的に***プロダクトライン***の実践を学んでいただくため，第1章でも登場した電子ポットの商品群を取り上げ，***プロダクトライン***の実際を身近に感じてもらうことにします．

電子ポットの題材はSESSAME（組込みソフトウェア管理者・技術者育成研究会）のワーキンググループ2が組み込みソフトウェア技術者の教育演習のために取り上げたもので，その設計のための詳細が記された仕様書は，SESSAMEのWebページで公開されています．本特集では，SESSAMEがまとめた電子ポットの仕様を「普及型」と位置づけ，近未来と未来型の電子ポットを追加して電子ポット商品群を構成し，この商品群について***プロダクトライン***分析を行っていきます．まずは電子ポット商品群のロードマップをご覧ください（**図3**）．

本稿で企画する普及型電子ポットG-2000は，SESSAMEが公開している電子ポットの基本機能（湯沸かし，保温，注ぎ，ミルクモード，キッチンタイマなど）に加え，後述する仮想の規制ロードマップや顧客満足度から展開したロードマップからきた節電機能，各種お茶モードなどの機能も追加しています．また，近未来型電子ポットG-7000や，未来型電子ポットG-9000では，最近話題になっている一人暮らしの老人の安否を電子ポットの使用状況からリモートで知る機能や，その機能を拡張し電子ポットが家電ネットワークに接続されることを想定して電子ポット以外の家電製品の使用状況についても地方自治体のケースワーカーや離れて暮らしている一人暮らしの老人の家族に知らせることができるような商品を想定しています．仕様の詳細と分析結果については，第3章で詳しく解説します．

商品の開発はG-2000からG-9000まで，3年間で五月雨式に行うこととし，3機種共通のコア資産の再利用は，この3機種にとどまらず，約5年間利用できるように作り込んでいくこと

〔図3〕電子ポット商品群のロードマップ例

〔図4〕電子ポット商品群に対する規制ロードマップ

（注：法律は架空のもの）

〔図5〕規制ロードマップを展開した技術ロードマップ

〔図6〕顧客満足度の観点から展開した機能ロードマップ

にします．

組み込み機器商品における国際規格や国内規格，各種規制はつきもので，商品をグローバルなマーケットで展開する必要のある企業にとって，クリアしなければならない第一のハードルです．また，商品の安全性や信頼性を確保するための規格以外にも，企業としての品質管理能力を客観的に示すためのISO9001や環境に配慮した取り組みを行っていることを示すISO14001なども，考慮する必要があります．次に**図4**の電子ポット（家電製品）に対する規制ロードマップ（これは仮想の規制．念のため）をご覧ください．ここでは，ISO9001の中で要求されているリスクマネージメントと，ISO14001に関連して想定した仮想の家電製品に対する節電規制と廃棄物処理規制を取り上げています．節電規制は，電子ポットを使用していないときの電力を最小限に抑えることが目的で，廃棄物処理規制は，電子ポットで使用しているモータやヒータなどの各ハードウェアデバイスをできるだけ長持ちさせ，かつ，自己診断機能により消耗したデバイスを特定し交換することによって，いたずらに家電ゴミを増やさないことをめざしています．**図5**は，このような仮想規制ロードマップから展開した要求技術のロードマップです．

安全機能強化は電子ポットの使用者の安全を確保するための異常監視機構であり，節電機能は単純なタイムアウト付きのタイマではなく電子ポットの使用状況を把握し使わない時間帯を予測して節電を行うインテリジェントな節電をめざします．また，電子ポットを長く使ってもらうためにハードウェアデバイスの使用頻度のログを記憶したり，自己診断機能を強化したり，消耗しているデバイスや故障しているデバイスを特定し交換できるようにします．

図6は，顧客満足度の観点から展開した機能ロードマップです．前述の規制ロードマップは商品として当然クリアされているべき機能ですが，それだけではグローバルなマーケットの競争に勝つことはできません．同じような商品仕様で勝ち残るためには，顧客満足度を重視したロードマップが必要です．**図6**では，電子ポット商品群の顧客満足度を高めるために「各種お茶モード」，「一人暮らしの老人の安否確認機能」，「ユビキタス時代を想定した家電ネットワーク機能」を想定したロードマップを描いています．各種お茶モードでは，レギュラーコーヒーの香りを引き立たせるようなお湯の注ぎ方や，日本茶やウーロン茶を入れるときのお湯の温度や注ぎ方のバリエーションの選択を可能にし，一人暮らしの老人の安否確認機能では，単純な安否確認から携帯電話機能を電子ポットにビルトインし，安否を確認したい老人の家族からの情報を電子ポットに内蔵したカラー液晶に表示する機能を追加し，家電ネットワーク機能では来たるべきユビキタスの時代に対応するために家庭内LANに接続し，ネットワークにつながれたすべての家電製品の使用状況を，電子ポットから安否確認を行うユーザーに発信する機能を付加します．

このような「商品群に対する規制ロードマップ」，「規制ロードマップから展開した技術ロードマップ」，「顧客満足度から展開したロードマップ」をふまえて**図3**の「電子ポット商品群のロードマップ」を描くことで，より具体的な商品群の未来像を浮き彫りにしていきます．

2）ドメイン構造図

ロードマップを描くことができたら，次はそれらの商品を実現するための機能を小さなドメインに分け，その小ドメイン同士の依存関係をドメイン構造図として描くことによって，再利用する単位を明確にする作業を行います．電子ポット商品群のドメイン構造図を作る具体的な過程については，次の第3章を参照してください．Appendixの**図6**および**図7**に，完成した普及型電子ポットG-2000のドメイン構造図と近未来型電子ポットG-7000のドメイン構造図を示します．両方のドメインに所属し電子ポット商品群のコア資産として再利用していくドメイン（パッケージ）は，太い枠になっています．このようにドメイン構造図からソフトウェアの再利用性が明確化できれば，これまでブラックボックスだったソフトウェアのモジュールも，明確な再利用の単位として扱うことが可能になります．

また，このようなロードマップからドメイン構造図への分析の中で，想定したコア資産を利用した各種のバリエーションのパターンは，実際のアプリケーションエンジニアリングの工程に対するヒントになり，魅力ある商品を作り出すための動機付けにもなります．

参考文献・URL

1）特別企画「品質と生産性向上のためのプロダクトライン入門」，『Software People』，vol.1，技術評論社
2）カーネギーメロン大学ソフトウェア工学研究所によるプロダクトラインのWebサイト http://www.sei.cmu.edu/plp/product_line_overview.html
3）EEBOFのWebサイト http://www.bof.jp/eebof/
4）ケント ベック著，長瀬嘉秀監訳，飯塚麻理香訳，永田渉訳，『XPエクストリーム・プログラミング入門』，ピアソン・エデュケーション
5）Paul Clements, Linda Northrop共著，前田卓雄訳，『ソフトウェアプロダクトライン――ユビキタスネットワーク時代のソフトウェアビジネス戦略と実践』，日刊工業新聞社

さかい・よしお　組込みソフトウェア管理者・技術者育成研究会(SESSAME)
いまぜき・たけし　(株)豆蔵

第3章 電子ポット商品群のドメイン構造図を作成する
コア資産を摘出するためのドメインエンジニアリングの実際

酒井由夫/今関　剛

第２章では，電子ポット商品群のロードマップを作成した．本章では，これをもとに，「ドメインエンジニアリング」を行い，電子ポット商品群のコア資産を摘出する．ドメインエンジニアリングとは，第２章でも説明したが，商品群の共通する機能や特性を摘出し，それらをまとめて再利用できる形式にする活動をいう．コア資産摘出のために，ドメイン構造図を書いていく．
（編集部）

はじめに

本章では，第2章で作成した電子ポット商品群のロードマップをもとに，***プロダクトライン***の三つの活動における**ドメインエンジニアリング**を行います．具体的には，電子ポット商品群の要求仕様を分析し，これらの商品に共通するコア資産を摘出します．コア資産を摘出する際には**UML**（**Unified Modeling Language**）のパッケージ図を使ってドメイン構造図を書きますが，UMLを使わなければいけないというわけではありません．ドメイン構造図は四角と矢印の描画，テキストの挿入，色づけさえ行うことができれば，一般的な図形描画ソフトでも作成することができます．本特集では，本章で作成した**ドメイン構造図**をもとにドメインに所属するクラスを作成し，クラス図やコラボレーション図を描いて最終的にはC++のコードを生成する作業を行うので，UMLによる分析から実装へのシームレスな連携を効率良く行うためにUMLツールを使用します．具体的には，第4章の「コア資産の設計・実装」で，UMLの各種図面を使いながら説明します．

1 普及型の電子ポットの要求仕様

普及型電子ポット「G-2000」の要求仕様は，組込みソフトウェア管理者・技術者育成研究会（SESSAME）がWebページ（http://www.sessame.jp/）でワーキンググループ2の成果物として公開しているものをベースにします．SESSAMEが公開している「話題沸騰ポットGOMA-1015」の要求仕様には，次のような電子ポットの基本機能が記述されています．

① ポット内の水を沸騰させ保温する機能
② ポット内の湯を給湯する機能
③ 指定した時間がきたらブザーを鳴らして知らせるキッチンタイマ機能
④ 保温設定の変更（高温：98℃，節約：90℃，ミルク：60℃）
⑤ ポット内のお湯の残り水位の表示

本稿ではこれらの基本機能に次の機能を追加し，普及型電子ポットG-2000を開発します．

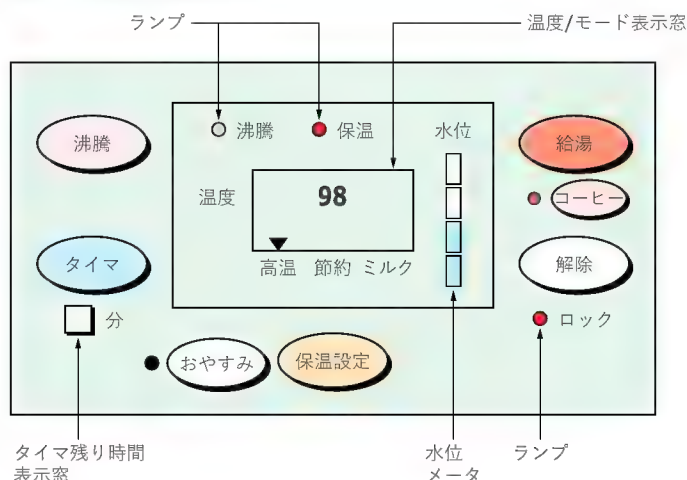
⑥ おやすみ機能（電子ポットが使われない時間帯を予測してヒータをオフにし電気を節約する機能）
⑦ コーヒー注ぎモード（レギュラーコーヒーにお湯を注ぐときにチョロチョロとお湯を注ぎ，香りを立たせる注ぎ方をする）

図1にG-2000の操作パネル例を示します．

また，次の第4章（電子ポットのコア資産の実装）で解説しますが，電子ポットに関するリスク分析の対策を実施するために，障害の監視機能を実装します．リスク分析と障害の監視機能は，組み込み機器にとって重要な要素なので，必ず実装すべき対象として認識するようにしてください．

2 近未来型の電子ポットの要求仕様

近未来型の電子ポット「G-7000」を買ってもらう顧客のターゲットは，一人暮らしのお年寄りとその家族です．実際に電子ポットの付加機能として一人暮らしのお年寄りが電子ポットを使用した状況を携帯電話にメールで知らせることができるサー

〔図1〕G-2000の操作パネル

ビスが実施されていますが，G-7000ではこの機能をさらに進化させ，携帯電話の機能そのものを電子ポットに内蔵させてしまうことを考えます．G-7000のカタログ案を**図2**に示します．

G-7000では，携帯電話などの電子機器の操作が苦手なお年寄りの手を煩わせることなく，お年寄りの息子や娘，孫，またケースワーカーなどとお年寄りがビジュアルなコミュニケーションを取ることを可能にします．息子や娘，孫がG-7000を使って知らせたいことがあるときは，送り手側はカメラ付きの携帯電話で画像メールを電子ポットに内蔵された携帯電話カードに送信し，お年寄りがお湯を給湯したりして電子ポットの操作を行うと，電子ポットの側面に配置されたカラー液晶画面に写真付きメールの内容が表示されて連絡があったのを知ることができます．カタログでは，孫が大学を卒業したことを知らせる写真付きメールを娘から受け取ったお年寄りが，娘や孫と話しをするために電子ポットからボタン一つであらかじめ記憶されている電話番号に電話をかけるという状況を説明しています．

電子ポットが記憶しておくべき電話番号やさまざまな設定項目は電子ポットに挿入するメモリカードから読み込まれるため，お年寄りの息子や娘はあらかじめ自分たちのパソコンなどで各種の設定をメモリカードに記憶させておくことが可能です．そのメモリカードを郵送し，お年寄りが電子ポットのメモリカードを差し込むことで，電子機器の操作が苦手なお年寄りにも簡単にカメラ付き携帯電話の機能を電子ポットを介して使うことができるようになります．さらに，高速なデータ通信が可能な次世代携帯電話のPCカードを使えば，音声での通話だけでなく動画を使ったテレビ電話の機能もG-7000で実現することが可能です．

電子ポットに写真付き携帯電話の機能を組み込むことなどせずに，TV電話や携帯電話をお年寄りに買ってあげればいいのではないかと感じられるかもしれませんが，次のようなお年寄りを取り巻く環境を考えれば，電子ポットに携帯電話機能を内蔵させることの有効性を理解していただけると思います．

① お年寄りに対して日常使う電子ポットにちょっとだけ付加価値を付けたという設定にして，大げさなことを考えているのではないという印象にする

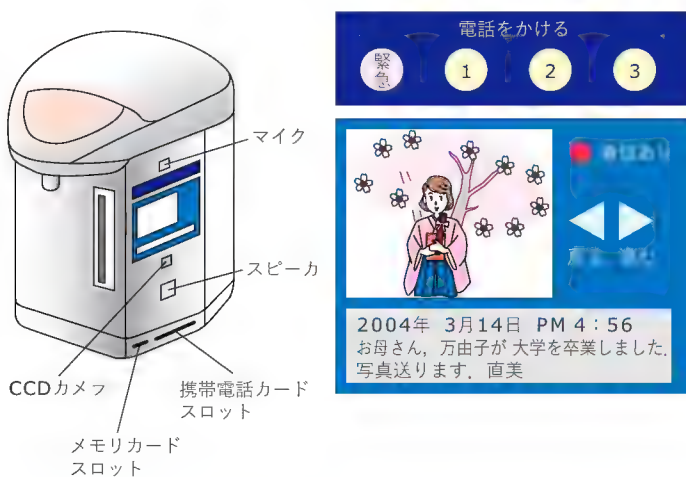
② 電話のようなライフラインは使い慣れているものをそのまま使ってもらい，新しい機能は付加価値として実現させ，万が一のときは，いつものやり方で連絡が取れるという安心感を大事にする

③ 息子や娘が親に電子ポットを贈り，より，密度の濃いコミュニケーションを取りたいという意図を伝える

④ 電子ポットの使用状況を知ることで，お年寄りに意識させずに安否を確認する

G-7000はあくまでも架空の商品ですが，実際に商品化するつもりでマーケティングを考えれば，このように一人暮らしのお年寄りの娘や息子のほかにも，地方自治体が一人暮らしのお年寄りの安否確認のためにG-7000を大量に購入するという状況もあるかもしれません．G-7000の目標販売価格は98,000円ですが，地方自治体からの100台単位の発注があれば1台あたり78,000円くらいに値引いてもいいでしょう．

〔図2〕G-7000のカタログ

電子ポットにカメラ付き携帯電話機能を内蔵！
これで一人暮らしの父・母も安心！
茶の間からいつでも連絡がとれます．

G-7000の特長

1. 携帯電話カードを内蔵させることにより，動画や静止画および電子メールを着信できる
2. 緊急時や，特定の三つの電話番号へボタン一つで電話をかけることができる
3. 電話番号登録者がポットの使用状況を自動で問い合わせることができる
4. 登録電話番号などの装置のセットアップ情報やプログラムのアップデートが，メモリカードですべて可能（パソコンで設定を行い，郵便で対象のお年寄りにメモリカードを送り，カードを差し込むことですべて設定を完了することが可能）

3 未来型の電子ポットの要求仕様

未来型電子ポットG-9000は，家電ネットワークが普及し，家庭電化製品同士がネットワークにより結合することを想定して，電子ポットと他の家庭電化製品が通信して家電製品全体の使用状況を外部へ通信し，お年寄りの安否確認をより正確なものにするということをめざしています．

具体的には，電子ポットよりも使用頻度の高い冷蔵庫や電子レンジ，電磁調理器，テレビなどの利用状況を電子ポットに内蔵された携帯電話PCカードを通じて取り出すという機能を追加します（第2章の商品ロードマップも参照．本稿ではG-9000のドメイン構造図は作成しない）．

4 ドメイン構造図とは？

本特集における商品群のコア資産を摘出するためのドメインエンジニアリングのフェーズでは，ドメイン構造図を書くこと

Column 1
アーキテクチャについて

もし，あなたが建設業界で仕事をしていると仮定しましょう．あなたがある橋の設計をまかされたら最初に検討すべきアーキテクチャ（建築様式・構造）とは何でしょうか？　橋に求められる基本的な機能は，河川，海峡，他の交通路などの上をまたいで道路や鉄道などを通すことであり，橋を架けたことによる効果として人や物資の流通が効率化されます．これを実現するために必要となる建造物が橋梁というわけです．さて，橋梁を建造する前に，いちばん初めに決めておかなければならないことがありますが，それが何かわかりますか？

それは予想される交通量に対する橋の強度のキャパシティや建造時に用いる材料，架設する距離や架設方法，景観などの条件のトレードオフにより，吊り橋，アーチ橋，桁橋，ラーメン橋などのさまざまな懸架方式の中から，もっとも適した懸架方式をまず選択することです．

このように，橋を作る具体的な作業に先立ち，事前に決定しておく設計の方針をアーキテクチャ（アルキ・テクトン：第一の技術/神の技術）と呼びます．アーキテクチャは，その後の具体的な設計を行うための指針として定められるため，設計途中での変更は基本的に行いません．また，アーキテクチャは橋を懸架する方式を説明する際の概念であり，具体的な実体（設計内容，建造物）ではないことに注意が必要です．

組み込みソフトウェア開発でも，状況的には橋の建造と同様のことがいえます．ソフトウェアにおけるアーキテクチャは，システムを支える機構，方式，構築上の指針となる技術的概念です．アーキテクチャの決定は，システムの構造，ふるまい，その使い方，機能，実行パフォーマンス，柔軟性（弾力性），再利用性，理解容易性，経済性，技術上の制約，美的観点，以上のトレードオフに関係してきます．また，既存フレームワーク候補の選択や，候補がない場合にはフレームワークを構築する決断にも影響します．

通常，組み込みソフトウェアは，メカやエレキなどと連携して開発を行います．したがって，メカ，エレキ，ソフト全体をひっくるめた製品のシステムに対するアーキテクチャ（システムアーキテクチャ）があります．この中で，メカ，エレキ，ソフトのそれぞれの役割分担を行い，分担した範囲でソフト内部のアーキテクチャ（ソフトウェアアーキテクチャ）を選択する必要があります．ただし，橋梁を建造する場合と違い，ソフトウェアに対するアーキテクチャは簡単に選択できるほどノウハウがまとまってはいないのが現実です（アーキテクチャパターンとしていくつか提案されている）．また，表現の手段についても厳格に定義されているものではありません．ただし一ついえることは，バランスのとれたさまざまな観点で，開発しようとするソフトウェア内部のコンセプトを関係者に納得してもらえることが必要であると考えます．

少々難しい説明になってしまいましたが，皆さんの開発現場を振り返ってみましょう．幸いなことに組み込みソフトウェア開発では，既存成果物の再利用を繰り返すことが多いという特徴をもっているので，まったくの新製品を立ち上げる場合を除き，毎回の開発において劇的にアーキテクチャが変更になるということは考えられません．したがって現状の開発では，ドメイン分析などの成果を反映したアーキテクチャを厳密に定義し，これをもとにフレームワークを構築し，大規模な再利用を実現することにより，ソフトウェア開発の生産性を向上させる余地がおおいにあるといえるのです．

によって電子ポットの機能をブロック分割し，それぞれの機能ブロックが最小の依存関係になるように，また，電子ポット商品群間で再利用するブロックとそれぞれの機種でしか利用できないブロックを明確にしていきます．このように電子ポット商品群の機能を分析し再利用性を検討することは，ロードマップに基づいた体系的な再利用を実現するためにはとても重要な作業です．

本特集の中では，この再利用の機能ブロックの単位を（狭義の）ドメインと考え，ドメインの構造と依存関係を表した図をドメイン構造図と呼ぶことにします．UMLのパッケージ図を用いてドメインの構造を表す考え方は，（株）オージス総研の渡辺氏らが書かれた『組み込みUML～eUMLによるオブジェクト指向組み込みシステム開発』[3] にくわしい解説があるので，そちらもぜひお読みください．

本稿で検討する電子ポット商品群は，静的制約や動的特性においてドラスティックな変化がないものとしてドメイン分割を行っています．このように分割された各ドメインは，静的制約や動的特性においてドラスティックな変化がないという条件下で再利用性の高いものとなり，用途の多様化や生産性向上を大きな課題としている組織にとって，アプリケーションエンジニアリングにおけるバリエーションが組みやすくなります．しかしながら，組み込みシステムにおける静的制約/動的特性/用途の多様化/生産性向上のバランスによっては，視点を変えてドメイン分割の方針を変えたほうがよい場合もあるということを，頭の片隅に残しておいてください．

5 最初のドメイン構造図の作成

まずは，完成した普及型電子ポットのG-2000のドメイン構造図と，近未来型のG-7000のドメイン構造図を見てください（Appendixの**図6**「G-2000のドメイン構造図」，**図7**「G-7000のドメイン構造図」）．

まとまった機能を一つのドメインとし，ドメイン同士の依存関係を点線の矢印によって表します．ドメイン構造図のトップ

は，装置そのものの全体機能を表すシンボル的なドメインとなります．装置の個々の機能は，さまざまなサブシステムが結合することによって実現されます．ドメイン構造図を書くことによってこのサブシステムとなる再利用可能なドメインを摘出し，分割することになります．組み込みシステムの場合はすでにリリースされた現行品のモデルがあり，商品のターゲットとなる市場や要求仕様があらかじめわかっている場合が多いので，この作業は比較的容易に行えると思います．しかし，作業が容易だからといって，現行製品の機能をただ単に分割してドメイン構造図にすればいいのではなく，再利用の観点からどのように機能分割すべきかを考えながらドメイン構造図を書いてください．その際には，第2章で作成した商品群のロードマップで商品群共通の機能をつかみ，個々の商品の要求仕様で商品別の特徴を把握するようにします．

ドメイン構造図を作成するにあたって注意すべきポイントは，ユーザーインターフェースのドメインの存在です．ユーザーインターフェースは商品の価値が顧客に最初に判断される「顔」の部分ですが，商品を開発するたびにユーザーが触れる部分を変化させていくようなコンシューマプロダクツでは，ユーザーインターフェースを実現するドメインのソフトウェアは再利用しにくい部分です．

ユーザーインターフェースがどんなに巨大であっても，再利用性の観点から見ると，商品群としての重要度・優先度が低いことがあります．このような場合は，ユーザーインターフェースドメインと再利用可能なドメインの位置づけを明確に分離して考えたほうが開発をスムーズに進めることができます．要求仕様が変化しやすいユーザーインターフェースドメインにだけ的を絞ってXP（eXtreme Programming）を適用するのもいいかもしれません．

6 ドメインを摘出する

ドメイン構造図を書くために，まず，システム全体のパッケージをいちばん上に配置し，ユーザーインターフェースをもった機器であればユーザーインターフェースドメインのパッケージを下のほうに配置し，その間に，その機器で実現する機能に関するドメインを配置していきます．ドメイン構造図を描き切るには，思いつくままに機能ブロックをドメインとして書き出し，依存関係を結んでみることです．UML表記法を使うのであれば，まず，ユースケース図を描いて商品の要求分析を行い，分析した要求から機能を抽出する方法もありますが，ここではユースケースの作成は飛ばして，普及型の電子ポットの要求仕様と近未来型の電子ポットの要求仕様からドメイン構造図を描いています．UMLで書いたユースケース図については第4章（電子ポットのコア資産の実装）で解説するので，そちらをご覧ください．

ドメイン構造図を構成するドメインの摘出は，オブジェクト指向設計のクラスの摘出に似ていますが，***プロダクトライン***におけるドメインエンジニアリングでは，再利用の単位としてのドメインを抽出するようにします．組み込みシステムにおけるドメイン構造図作成の簡単な方法は，極端にいえば機器で実現する機能ごとに「○○アプリケーション」と「○○（ハードがらみ）」といったドメインを作ってしまうことです．たとえば，電子ポットの普及型モデルであるG-2000の場合，次のようなドメインを作ることができます（**図3**）．

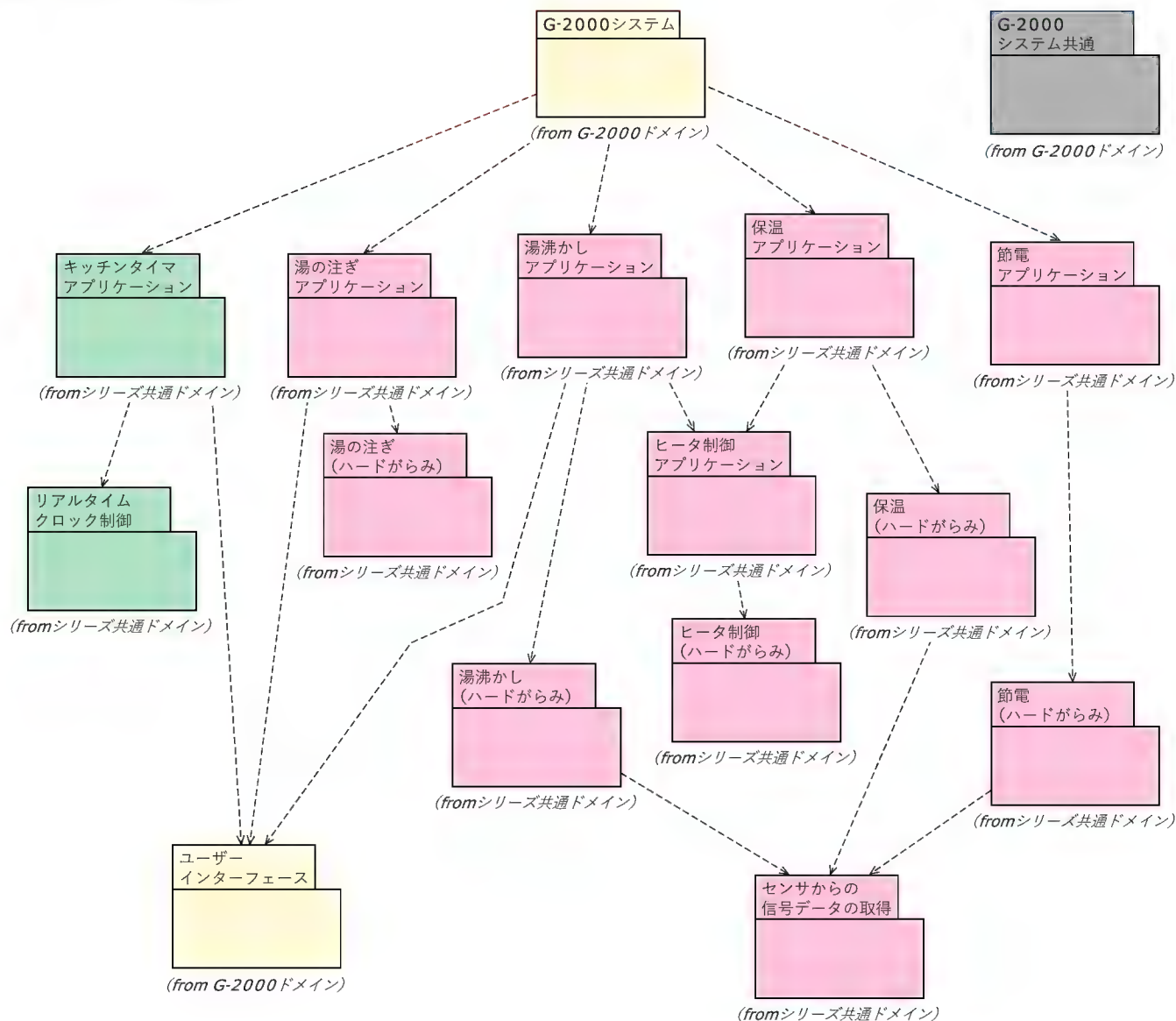
① 「湯沸かしアプリケーション」，「湯沸かし（ハードがらみ）」
② 「湯の注ぎアプリケーション」，「湯の注ぎ（ハードがらみ）」
③ 「保温アプリケーション」，「保温（ハードがらみ）」
④ 「節電アプリケーション」，「節電（ハードがらみ）」
⑤ 「キッチンタイマアプリケーション」，「キッチンタイマ（ハードがらみ）」

組み込み機器ではハードウェアを使ってある機能を実現することが多いので，「○○アプリケーション」と「○○（ハードがらみ）」という2種類のドメインをペアで作ります．これは○○という機能を実現するためのドメインをハードウェアに関係する部分とソフトウェアのみで独立できる部分に分けて，「○○アプリケーション」ドメインの再利用性を高めるためです．ハードウェアに関係するドメインは，ハードウェアが変更されると修正を余儀なくされることが多いので，ソフトウェアだけで独立できるドメインを分けて再利用性を高めることを考えます．

機能別にドメインを書き出して依存関係の線をつないだら，もう一度商品の要求仕様を読み直して，作成したドメイン構造図で要求仕様をすべて実現できるかどうかを検証してみます．足りない機能があったら，ドメインを追加してすべての機能が実現できるようにします．このような漏れや抜けを防ぐためには，第4章の冒頭で説明するように，ユースケースで要求を分析し必要なハードウェアデバイスをサブアクタとして書き出しておく方法も有効です〔くわしくは参考文献4）を参照のこと〕．

この時点でのドメイン同士の依存関係を表す矢印の方向については，それほど気にする必要はありません．ドメイン間の依存関係はこの後，ドメイン構造図をプロジェクトメンバでレビューしたり，ドメインの中身を実装していく過程でドメイン同士の関係を見直したりすることで，必ず修正が入ります．ドメイン構造図は何回も修正することで完成度を高めることに意味があるのです．

図3の例では，レビューやディスカッションの後に「湯沸かし（ハードがらみ）」と「保温（ハードがらみ）」と「節電（ハードがらみ）」はすべてヒータをコントロールするドメインなので共通化できることがわかり，「ヒータ制御（ハードがらみ）」というドメインに統合されました．また，「キッチンタイマ（ハードがらみ）」は実際にはリアルタイムクロックの制御について責務をもつドメインであることがわかったので，「リアルタイムクロックの制御」という名前のドメインになりました．このように，ドメイン構造図におけるドメインの分割や統合・合併は，レ

〔図3〕初期のドメイン構造図

ビューが行われるたびに発生します．これは，決して無駄な作業ではなく，再利用可能なコア資産の抽象度を上げるためのたいせつなプロセスです．ドメイン構造図を洗練させる工程なくして，最初から抽象度の高いモデルを作ることは難しいと考えられます．そのことを十分に考慮して，ドメイン構造図は何回も書き直すことができるドローイングツールやUMLツールを使用されることをおすすめします．

7 近未来型電子ポット「G-7000」におけるドメインの摘出について

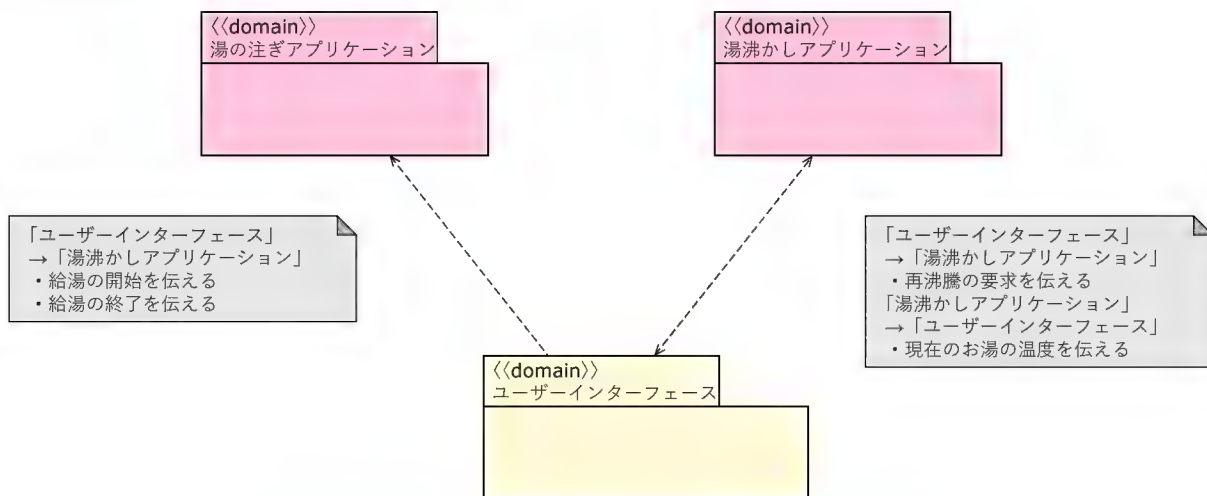
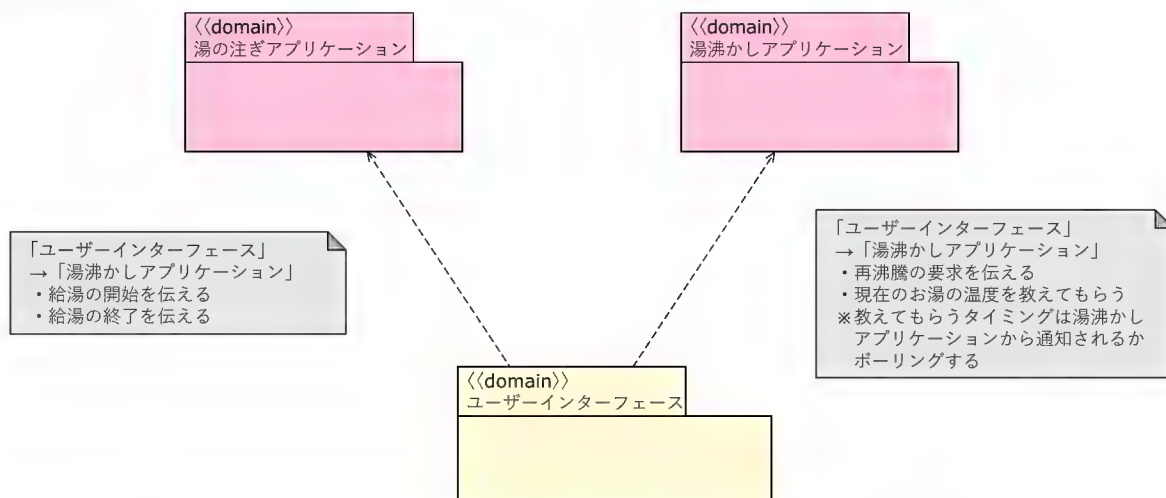
G-7000では，G-2000で利用したユーザーインターフェース以外のすべてのドメイン（共通コア資産）に，携帯電話カード関連やメモリカードの制御，安否確認アプリケーション，バッテリの制御などのドメインを追加して，ドメイン構造図を完成させます．G-7000のユーザーインターフェースドメインと新たにドメインに関連する依存関係を記入します．G-2000のドメイン構造図からもってきた共通資産となり得るドメイン群と新たに追加したドメインは基本的には独立していますが，G-7000で追加した「安否確認アプリケーションドメイン」で，操作者が給湯や沸騰の操作を行ったことを知る必要があるので「センサからの信号データの取得ドメイン」と「安否確認アプリケーション」の間に依存関係が引かれることになります（参照：Appendixの**図7**「G-7000のドメイン構造図」）．

8 ドメイン間の依存関係はどう示すか？

ドメイン間の依存関係は，点線の矢印で表します．依存関係の基本的な考え方は，矢印の先にあるドメインが存在しなければ矢印のもとにあるドメインの機能を実現できないという関係を表します．ドメイン同士が相互に依存しているということは，二つのドメインの結合度が強いということですから，できるだけ矢印が片方だけにつくようにします．Appendix の**図 6**「G-2000 のドメイン構造図」を見てもらえばわかるように，ユーザーインターフェースドメインは，操作者と組み込み機器の間の窓口であるため，いろいろなドメインとの関係が集中しやすいという特徴があります．

ユーザーインターフェースドメインは，操作者の意図をキーやスイッチを通して各ドメインに伝える入力系の役割と，各ドメインが行った仕事の結果を操作者に伝えるための出力系の役割の二つの役割をもっています．したがって，ユーザーインターフェースドメインと各ドメインとの依存関係を表す矢印の向きは，多くの場合，入力系の役割でユーザーインターフェースドメインが使われる場合はユーザーインターフェースドメインから出る方向へ，出力系の役割ではユーザーインターフェースドメインに入る方向になります．

たとえば，給湯ボタンを押してお湯を注ぐ場合は，ボタンを管理しているユーザーインターフェースから湯の注ぎアプリケーションドメインへ依存の矢印が向かいます．一方，現在のお湯の温度情報は湯沸かしアプリケーションドメインからユーザーインターフェースドメインに伝えられるので，矢印の向きは湯沸かしアプリケーションドメインからユーザーインターフェースドメインの方向になります．しかし，再沸騰ボタンを押して再沸騰を行う行為は，ユーザーインターフェースドメインから湯沸かしアプリケーションへの向きになるので，お湯の温度の表示と再沸騰を合わせるとユーザーインターフェースド

〔図 4〕相互依存の関係

〔図 5〕相互依存の解消

メインと湯沸かしドメインは相互依存の関係になってしまいます(**図4**).

再利用性を考えた場合，湯沸かしアプリケーションドメインは，他のドメインからパッシブ(受け身)であるほうが有利です．したがってお湯の現在温度は，ユーザーインターフェースドメインが表示に必要な周期で湯沸かしアプリケーションドメインに現在のお湯の温度を聞きにいくか(ポーリング)，お湯の温度が変化したときに，そのタイミングだけを湯沸かしアプリケーションドメインからもらって，湯沸かしアプリケーションドメインへ温度情報を聞きにいくといった操作にしたほうが良いでしょう(**図5**).

温度が変化したタイミングを知らせてもらうことを，湯沸かしアプリケーションドメインからユーザーインターフェースドメインへの依存ととらえることもできますが，タイミングを知らせてもらうだけであれば，依存の矢印にはならないとする考え方もあります．本稿では，後者の考え方をとり，依存の矢印はユーザーインターフェースドメインから湯沸かしアプリケーションドメインへの一方向であると考えます．なお，ソフトウェアを実装する際に，このような依存関係を一方向にするためのオブジェクト指向設計におけるデザインパターン(オブザーバパターン)を用いることで，ドメイン間の独立性を高く見せるテクニックがあります．

Column 3
デザインパターンとは？

ソフトウェア開発におけるオブジェクト指向技術の重要性が高まることにより，ソフトウェア開発上必要となる情報を表現する手段として，UMLが普及してきました．しかし依然として，ソフトウェアの内部設計を効率良く理解したり，他人とコミュニケーションしたりする手段は不十分といえるでしょう．ベテラン技術者の頭の中が自由に覗けたらいいなと思ったことはありませんか？　設計の意図や根拠を共有する手段はないのでしょうか？この点に関して一つのカギを握る手段がデザインパターンです．

そもそも，デザインパターンとそれを抽象化し整理したメタパターン，フレームワーク技術や分散オブジェクト技術などの手法や考え方は，実開発におけるオブジェクト指向の適用経験から生まれてきました．この中でもとくにフレームワークは，開発生産性と再利用可能性を高めるために，必要に応じて取り替え可能な機能と，スケーラビリティ性のある性能を，固定的な多くの制御構造とともに提供する必要があります．このため，これらの機能および性能の可変部分はアドホック(場当たり的)に構築されるものではなく，あらかじめ想定したソフトウェア群に対し，拡張性および保守性の確保を一定の期間保証するものとして構築されます．しかし，開発対象そのものが複雑かつ大規模なため，何もないところからこのようなしくみ/枠組みを構築していくには，相当な開発スキルと工数を要します．したがって今日では，デザインパターンを組み合わせることが，フレームワークを効率的/効果的に構築/構成するためのたいへん有効な手段の一つとなっています．

デザインパターンには，あらかじめ解決すべき課題とそれに対する実現手段が定義されています．一見，あるパターンの内部構造/メカニズムが他のパターンと類似していたとしても，それぞれが得意とする解決対象範囲は異なり，パターン名も異なったものが用いられます．したがって，フレームワークを構築する際にデザインパターンを適用すると，解決すべき課題/開発担当者の設計意図/パターン選択の根拠が明確化し，フレームワーク利用者にとって理解しやすいものとなります．また，フレームワーク全体のアーキテクチャを理解する際，パターン名を用いてコミュニケーションすることで細かな構造やふるまいを詳細に説明する手間を省くことができるのです．さらに，機能拡張方針などの理解にも役立ち，結果として開発生産性を向上させることが可能となります．

ソフトウェアの分野でデザインパターンを初めて提唱したのは，E.Gamma, R.Helm, R.Johnson, J.VlissidesらGoF(the Gang of Four)でした．GoFによるデザインパターンは，GUIベースのシステムを構築する際に獲得した設計ノウハウをカタログしたものであったため，今日の広範囲なソフトウェア開発ドメインの視点に立つと，狭義のデザインパターンと認識されています．

しかし，GoFが取り上げたものも含め，デザインパターンが特定のアプリケーションに特化せず，適度な抽象度を保持していることから，これらノウハウとして蓄積されてきた成果は，現在のソフトウェア設計現場に徐々に取り入れられています．ひょっとして読者の方の中には，デザインパターンに関する文献を参照していて，同じようなパターンを既存の設計に適用している経験があると感じる方がいるかもしれません．優れた技術者であれば，さまざまなトレードオフを検討した結果，似通ったパターンを適用してフレームワークを無意識に構成していたということがあるかもしれません．

デザインパターンに対する開発者のとらえ方は千差万別です．デザインパターンの考え方を臨機応変に適用できる技術者は問題ありません．しかし，本来クリエイティブで自由な発想でブレークスルーしなければならないところを，中にはパターンを厳格に適用することにこだわり，パターン自体に設計がしばられてしまう開発者をときたま見受けます．実開発での適用局面において，GoFデザインパターンすべてを知っている必要はありません．また，GoFデザインパターン以外のパターン(P.COADのパターンなど)を開発に適用してはいけないというルールもありません．場合によっては，デザインパターンを自分で作り出して命名してもかまいません(開発関係者やプロジェクトおよび組織内で周知していく必要があるが)．実開発においてデザインパターンを理解することや適用することが目的となってしまっては本末転倒です．優れた開発者にとって，現在業界で流通しているパターンをソフトウェア開発に柔軟に採り入れ，ソフトウェアの生産性と品質の向上に役立てるスキルを身につけることが肝要です．

ユーザーインターフェースは，組み込み機器によってはシステム全体の大部分を占めることがあります．しかし，再利用の観点から見るとユーザーインターフェースは機種間で変化しやすいドメインなので，電子ポット商品群ではドメインの規模が大きいからといってユーザーインターフェースを詳細に機能分割することは，今回はドメインエンジニアリングの工程では行いません．ドメインエンジニアリングのフェーズで興味があるのは，「そのドメインがどれくらい再利用性の高いドメインかどうか」です．規模が小さくても再利用率の高いドメインは，ドメインエンジニアリングの中では価値が高く，コア資産となり得る重要なモジュールとなります．

9 ドメイン構造図でコア資産を表す！

ドメイン構造図では再利用率の高さを示すために，分割した機能ドメインのパッケージの太さで再利用率の高さを示すことにします．また，機能のグルーピングはパッケージに色を付けることによって表現します．これによって，商品の機能が再利用のドメイン単位として表現され，どのドメインとドメインがグループになっていて，コア資産としての重要度がどれくらいあるのかを直感的に把握することができます．また，普及型の電子ポットのドメイン構造図と近未来型または未来型の電子ポットのドメイン構造図を並べて見ることによって，どのドメイン構造図にも現れるドメインが，商品群としてのコア資産となる重要なモジュールであることを認識できます（参照：Appendixの**図6**「G-2000のドメイン構造図」，**図7**「G-7000のドメイン構造図」）．

また，二つ以上の異なる商品のモジュール構造図を並べてそれぞれの商品の動作をレビューすることで，コア資産となるドメインの抽象度を高め再利用性を向上させるための修正作業（洗練）を行うことができます．ロードマップをもとに複数の商品のドメイン構造図を作り，「コア資産」と成り得るドメインを「摘出」し，それらの「コア資産」がそれぞれの商品の中で，どのように「利用」されるのかをレビューし，コア資産のあり方を見直して「フィードバック」するといった一連の作業は，***プロダクトライン***の（ドメインエンジニアリングにおけるコア資産の摘出）（アプリケーションエンジニアリングにおけるコア資産の利用）（コア資産のマネージメント活動におけるフィードバック）の三つの活動を実際に回していることにほかなりません（参照：第2章の**図2**「***プロダクトライン***の三つの活動」）．

もちろん，***プロダクトライン***開発におけるマネージメント活動は本来，アプリケーションエンジニアリングの工程でコア資産を実装する際に気がついた点を，ドメインエンジニアリングのチームにフィードバックしながらコア資産を洗練していくのですが，ロードマップからドメイン構造図を展開した分析工程でもこのように***プロダクトライン***の活動のサイクルを回すことは可能です（規模の小さいプロジェクトでは，アプリケーションエンジニアリングのチームとドメインエンジニアリングのチームは同一であることもある）．ポイントは，アプリケーションエンジニアリングでコア資産の再利用率が高まるように，コア資産とそれ以外のドメインの依存関係ができるだけ小さくなるような，ドメインの分割を行い，ドメイン間のインターフェースを定義することです．

● ドメイン構造図の利用方法

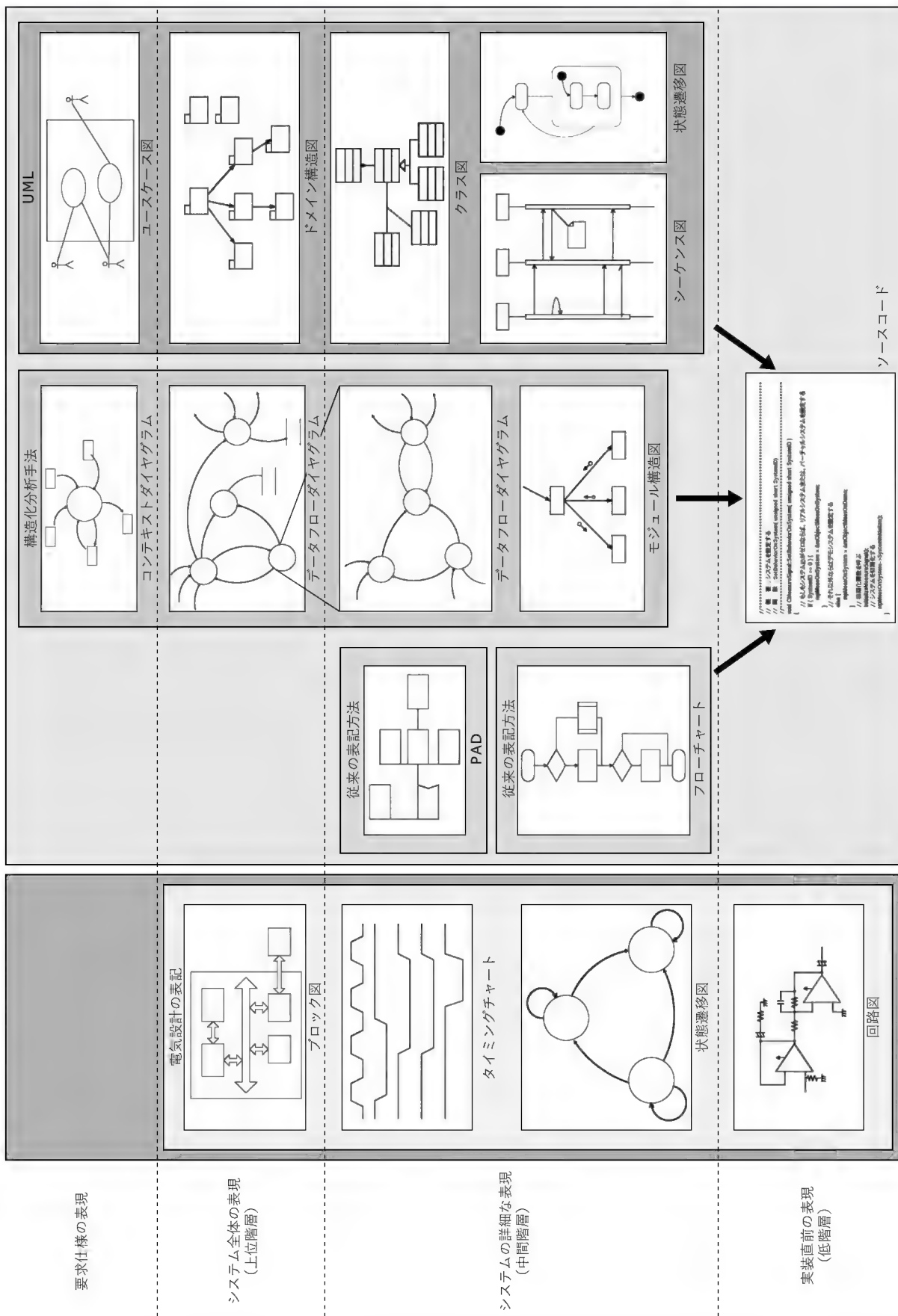
参考文献3)では，ドメイン構造図を書く作業をオブジェクト指向設計プロセス中の分析の部分に位置づけていますが，本稿ではドメイン構造図を書くフェーズを***プロダクトライン***の三つの活動（ドメインエンジニアリング，アプリケーションエンジニアリング，マネージメント：第2章の**図2**も参照）のドメインエンジニアリングの工程であると考えます．これは，組み込みシステム開発に置ける視点（スコープ）の違いです．実際に行っている分析作業自体に大きな違いはありません．

機能ブロックに見立てた（狭義の）ドメインを効果的に再利用するためには，ドメイン間の依存関係を最小にし，他のドメインに対するインターフェースを明確化し，ドメイン内部のアルゴリズムを隠ぺいするのが効果的です．このような，依存関係の最小化，インターフェースの明確化，内部アルゴリズムの隠ぺいといった設計方針は，オブジェクト指向設計向きであることは間違いありません．しかし，オブジェクト指向設計を実際に行うには，オブジェクト指向の考え方を理解し，エンジニアに専門教育を受けさせる必要があるので，簡単ではありません．***プロダクトライン***設計におけるドメインエンジニアリングは，オブジェクト指向設計を前提にした取り組みではないので，C++などのオブジェクト指向言語を使わなくてもドメイン構造図を描き，既存の商品群を分析し，再利用の単位を明確にすることで体系的な再利用活動を行うことは可能です．

ドメイン構造図を前にして開発チームのエンジニア全員が，自分の作成したクラスや関数がどのドメインに所属しているのかを意識しながらディスカッションすることは，組み込みシステムの再利用の単位としてのドメインを再認識し，ドメイン間の依存関係について考え直す絶好の機会です．実際にドメイン構造図を見ながらレビューを行うと，ドメイン間で相互依存していることが見つかったり，依存の方向性が逆だったりすることがわかります．このような修正を繰り返すことによって，プロジェクトメンバ全員が再利用の単位としてのドメインの認識とドメイン間の依存関係について学習できます．

また，組み込みソフトウェアの分析者やマネージャは，ロードマップに描かれた次期製品の仕様を頭に入れながらドメイン構造図をレビューすることで，商品群を意識したソフトウェア設計を成功させることができます．これまで組み込み商品群を開発する際にソフトウェアの全体構造を分析していなかったり，再利用性を意識したモジュール分割を行っていなかったりしていたのであれば，ドメイン構造図を書くことは大きな進歩です．設計レビューの際にディスカッションの対象となっているソフ

〔図6〕表記方法の分析

エレクトロニクス
エンジニアリングでの表記法
ソフトウェア
エンジニアリングでの表記法
要求仕様の表現
システム全体の表現
（上位階層）
システムの詳細な表現
（中間階層）
実装直前の表現
（低階層）
構造化分析手法
コンテキストダイヤグラム
データフローダイヤグラム
データフローダイヤグラム
モジュール構造図
UML
ユースケース図
ドメイン構造図
クラス図
シーケンス図
状態遷移図
電気設計の表記
ブロック図
タイミングチャート
状態遷移図
回路図
従来の表記方法
PAD
従来の表記方法
フローチャート
ソースコード

トウェアのモジュール(クラスや関数)がどのドメインに所属しているのかを確認しながらレビューを行うと，エンジニアの再利用に対する意識とドメイン自体の再利用性が高まります．

UMLを使った設計において一般的なドメイン構造図は，組み込みシステム全体の分析レベルから見ると，要求分析と実装設計の中間的な位置づけです．**図6**(前頁)の表記法の分類からもわかるように，ドメイン構造図は，UMLでいえば，ユースケースによる要求分析図とクラス図の中間に当たります．ユースケースからクラス図を直接描かずにドメイン構造図を入れてワンクッション置いた形です．ドメイン構造図は，最初に描いた図が開発の最後まで修正されずに使われることは，まず間違いなくありません．何回も何回も修正されるものだと考えてください．繰り返し修正するのは決して悪いことではなく，修正する行為がモデルを洗練し再利用性を高めることにつながると考えてください．このようなとき，UMLツールを利用していると，ドメイン構造図の修正やドメインに所属するクラスのドメイン間の移動が簡単に行うことができて便利です．

10 ドメインの入れ物に内容物を入れる

ドメイン構造図がある程度できあがったら，ドメインに所属するクラスや関数を作成していきます．この工程は，参考文献3)でいえばアーキテクトデザインのフェーズであり，***プロダクトライン***でいえばアプリケーションエンジニアリングの活動ということになります．

ドメイン構造図におけるドメインは，単なる入れ物です．入れ物に入れる中身は新規に作成してもよいし，すでに存在しているシステムのモジュールをクラスや関数レベルでもってくるという考え方でもまったくかまいません．すでにあるC言語で書かれた関数の固まりがあるのならば，それらの関数がどのドメインに所属するのかを振り分けていくだけでも，システム全体の構造が整理され，違うドメインに所属している関数同士が相互にメッセージを交換していたり，結合度が高かったりすることが明確になります．

このように，既存のクラスや関数をドメイン構造図の各ドメインに振り分け，ドメイン間のインターフェースを明確にしたり，結合が疎になるようにインターフェースを修正したりするだけでも，ソフトウェア資産の再利用性は向上します．

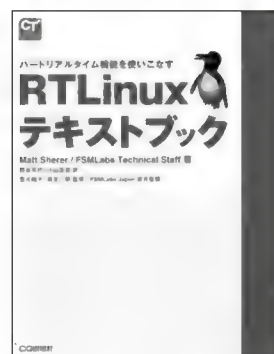
もしも，対象システムをオブジェクト指向設計で開発するのであれば，UMLツールを使うことによって，このようなクラス間のインターフェースの改善やクラスのドメイン間移動などを簡単に行えます．また，UMLツールを使ってリバースエンジニアリングを行えば，既存のC++やJavaのソースコードを読み込んでクラスを作り，ドメイン構造図で作成したドメインの入れ物に放り込めるので，もともとあるソースコードの帰属を気にすることなく，ドメイン分析をやり直すことができます．既存のソフトウェア資産を捨てることなく，徐々にソフトウェアモジュールの再利用性を高めることも，ドメイン構造図を書く利点の一つです．

また，既存のクラスや関数群をドメインにマッピングする段階で，ドメインの名前の見直しやドメインの統合・分割が必要になりますが，ドメイン構造図におけるドメインは単なる入れ物であり，ソフトウェアの実体ではないため，このようなドメインの統合・分割によって発生するクラスや関数の修正は最小で済みます．したがって，ドメイン構造図を洗練する作業は，ソフトウェアの実装フェーズに入ってからも積極的に行うことが可能です．

参考文献・URL

1) 組込みソフトウェア管理者・技術者育成研究会(SESSAME)，話題沸騰ポット要求仕様書 http://www.sessame.jp/workinggroup/WorkingGroup2/POT_Specification.htm
2) ケント・ベック著，長瀬嘉秀監訳，飯塚麻理香訳，永田渉訳，『XPエクストリーム・プログラミング入門』，ピアソン・エデュケーション
3) 渡辺博之，渡辺政彦，堀松和人，渡守武和記共著，『組み込みUML～eUMLによるオブジェクト指向組み込みシステム開発～』，翔泳社

さかい・よしお 組込みソフトウェア管理者・技術者育成研究会(SESSAME)
いまぜき・たけし (株)豆蔵

News Flash──東芝，手のひらサイズで1W出力の小型燃料電池を開発
(株)東芝は，携帯電話などのモバイル機器に使用できる小型の燃料電池を開発した．開発したのは出力1W，容積140cc，重量130gのもので，高濃度メタノールを発電時に発生した水で希釈しながら使用する「希釈循環システム」を採用しているのが特徴である．

第4章 再利用性を重視した実装方法
電子ポット商品群のコア資産をC++で実装する

酒井由夫/江藤善一/今関　剛

前の第3章では，電子ポット商品群のドメイン構造図を作成した．本章では，このドメイン構造図の各ドメインに入る内容物，つまり実装コードを作成する．そのため，UMLツールでクラス図を書き，クラス名やタスク分割の方法を検討する．そして，C++の実装コードを書いていく．ここで解説するソースコードとVisual C++のプロジェクトファイルは，本誌Webページからダウンロードできるようにし，簡単なテストも行えるようにする予定である．　　(編集部)

はじめに

本章では，第3章で作成したドメイン構造図のそれぞれのドメインに入れる内容物を作成します．ここではUMLツールを使ってクラス図を書き，C++で実装コードを書きますが，ひとまずはオブジェクト指向言語を使わずにドメイン構造図で問題領域の分割だけを行い，ドメインの内容物をC言語で書くというステップを踏んでもよいと思います．そのようなアプローチを取る場合は，本章を飛ばしていただいてもけっこうです．

しかし，分析のみならず実装工程にも体系的な再利用を導入しソフトウェア資産の利用率を上げたいのであれば，オブジェクト指向設計を実施することは有効です．本章のコア資産の実装工程では，オブジェクト指向設計を本格的に行ったことのないプロジェクトチームが，パイロットテーマや規模の小さい開発で試験的にオブジェクト指向設計を試すことができるように，安価で高機能なUMLツールの一つEnterprise Architect(**コラム1**参照)を使って電子ポット商品群のコア資産の実装を実現しています[注1]．

ソフトウェア設計・開発に強力なツールを導入するとツール依存になり，いつの間にかツールを使う目的を見失って使用者がツールに使われてしまうおそれがありますが，ツールを使うことのメリット・デメリットをきちんと認識して使えば，作成した図面類の再利用も可能になり，場合によってはソースコード生成の助けにもなるでしょう．

なお，オブジェクト指向の設計手法(たとえばユースケースの書き方やクラスの摘出の方法)にはさまざまな考え方があり，何をオブジェクトとしてとらえるのかによって，作成される図もおのずと変化していきます．本章で解説する方法もそのようなオブジェクト指向設計手法の一つであり，他の方法でも各ドメインの機能を実装できることを認識しておいてください．

1　普及型電子ポット「G-2000」の基本機能の確認から……

普及型電子ポット「G-2000」の基本機能をおさらいします．第3章の**図1**も参照してください[注2]．

- タイマボタン：このボタンを押すとタイマが起動し，1回押すごとに1分ずつ加算される
- タイマ残り時間表示窓：タイムアップまでの残り時間(分単位に切り上げ)が表示される
- 保温設定ボタン：保温モードを高温(98℃)/節約(90℃)/ミルク(60℃)モードに切り替える
- 温度/モード表示窓：現在の水温と設定されている保温モード(第3章**図1**の▼)が表示される
- 解除ボタン：給湯のロック/解除を行う．ロック中は給湯ボタンを押してもお湯は出ない．ロック中に押すとロックは解除され，解除されているときに押すと給湯をロックする．また，給湯中はロックできない
- ロックランプ：給湯がロックされているかどうかを表す．給湯がロックされているときに点灯する
- 給湯ボタン：このボタンを押すとポンプを動作させて給湯口からお湯を排出する．押している間は給湯を行い，ボタンから手を離すと給湯を停止する
- コーヒーボタン：このボタンを押すと，レギュラーコーヒーを入れるときにはお湯をチョロチョロと出して香りを立たせるモードになる
- コーヒーモードランプ：コーヒーモードのときに点灯し，通常の給湯モードのときに消灯する
- 沸騰ボタン：このボタンを押すと，ポット内の水を沸騰させてカルキ抜きを行う．沸騰中に押すと沸騰を中止して保温状態になる．1回押すごとに沸騰→保温→沸騰と変わる

注1：本特集で使用したUML各種図面とEnterprise Architectの30日限定特別評価版をInterface誌のWebページからダウンロードできるようにする予定．
注2：普及型電子ポットのG-2000要求仕様の詳細は，組込みソフトウェア管理者・技術者育成研究会(SESSAME)がWebサイト(http://www.sessame.jp/)でワーキンググループ2の成果物として公開しているものを基本にしている．

- 沸騰ランプ：お湯を沸かしているときに点灯する．沸騰が終了すると消灯する
- 保温ランプ：沸騰中でないときに点灯する．操作者が沸騰ボタンの押下などでお湯を沸かしはじめたときに消灯する
- 水位メータ：ポット内の水位を表示する
- おやすみボタン：このボタンを押すと，ポットを使わない時間帯は自動的にヒータをOFFにして電気代を節約する(**図1**)
- おやすみモードランプ：おやすみモードのときに点灯し，おやすみモードが解除されたときに消灯する

2 ユースケースの分析を行う

本特集では，組み込みソフトウェアシステムの分析・実装について，参考文献3)を参考にしています．よりくわしい解説を知りたい方は，参考文献3)もぜひご一読ください．

まず，電子ポット商品群の要求仕様をもとにユースケースを書きます．電子ポット商品群では，普及版の「G-2000」は一般消費者が対象の商品であり，「G-7000」と「G-9000」は一人暮らしのお年寄りをおもな対象としているので，ユースケースのアクタも変わってきます．G-2000とG-7000についてユースケースを書き，機能別に色分けをしてみます．完成したG-2000とG-7000のユースケースをご覧ください(Appendix **図1**，**図2**)赤が電子ポットとしての基本機能，緑が電子ポットとしての付加価値機能，青と黄色が上位機種独自の付加価値機能です．

次に，商品の要求仕様から必要な電子ポット内のデバイスをサブアクタとして抽出し，アクタとユースケースとの関連を書

Column (1)

安価で高機能な日本語UMLツール『Enterprise Architect』の紹介

UMLのドローイングツールとしてはラショナルソフトウェア(現IBM)のRoseが有名ですが，2003年の4月1日に安価で高機能なUMLツール「Enterprise Architect」の日本語版がスパークスシステムズ ジャパン(http://www.sparxsystems.co.jp/)より発売されました(**表A**)．Enterprise Architect日本語版はデスクトップ版，プロフェッショナル版，コーポレート版の3種類があり，デスクトップ版，プロフェッショナル版，コーポレート版の1ライセンスの価格はそれぞれ13,000円，21,000円，25,000円です．ほとんどの機能が含まれているプロフェッショナル版でもエンジニアがポケットマネーで購入できる程度の価格であり，5人くらいの小規模なプロジェクトなら割引が適用され総額95,000円で，社長などの決裁なしに5本も購入することができます．

本特集で紹介したような，体系的な再利用のための分析を実際にやってみようとした場合，レビューやディスカッションの後に図を繰り返し書き直す作業が発生します．このような図の書き直しを行う際に，優れたUMLドローイングツールがあると，図を書くことに多大な工数を取られることなくシステムの分析に集中できます．また，対象となるプロジェクトが初めてオブジェクト指向設計にトライしようとする場合，多額の予算を確保することは難しいため，UMLツールは安価に手配する必要があります．そのようなパイロットプロジェクトや小規模なプロジェクトでUMLやC++を使ってオブジェクト指向設計を試してみるのに，Enterprise Architectは最適なツールの一つといえます．

Enterprise Architect(プロフェッショナル版)は，**表1**のような特長を見ると，とても21,000円とは思えません．スパークスシステムズ ジャパンは個人や規模の小さい企業のユーザーにも利用可能なUMLのツールを提供しUMLのすそ野を広げていきたいというコンセプトのため，広告やカタログ・パンフレットの作成などの宣伝活動をあまり行わないことでこの価格を実現しているようです(メールやWebページでのサポートは充実している)．

本稿ではEnterprise ArchitectでUMLの各種図面を作成しており，スパークスシステムズ ジャパンの厚意によりEnterprise Architectの30日限定特別評価版(英語版ベースに日本語表示の初期設定がされているもの)を用意してもらいましたので，Enterprise Architectの特別評価版を本誌のWebページからダウンロードし，実際に本稿で使用したEnterprise Architectのプロジェクトファイルをブラウズしたりカスタマイズしたりして使い勝手を体感してみてください(本誌ダウンロードページ http://www.cqpub.co.jp/interface/download/contents.htm)．

〔表A〕Enterprise Architectのおもな特徴

●UML1.4に準拠しているすべてのUMLダイアグラムを描ける
●図やUMLの要素をWindowsのエクスプローラ風のインターフェースで見ることができ，個々のユースケース，クラス，アクタなど，さまざまなUMLの要素や図を，格納されているパッケージ(Windowsのフォルダに相当)から別のパッケージへ簡単に移動できる→レビュー後に図を容易に移動，修正，追加できる
●パッケージの間をUMLの要素群が移動しても，UMLの要素間の依存，継承，集約など関連情報は失われず引き継がれる
●C++，Java，C#，VB，VB .NET，Delphiの生成と読み込み(リバースエンジニアリング)が可能→すでにあるプログラムソースを解析したり，作成したUMLのクラスからスケルトンのソースコードを生成できる
●XML(UML1.3 XMI1.1)形式でのUMLモデル入出力ができる→出力したXMLを構成管理することでコンパクトな差分管理が可能
●HTMLやWordと互換性のあるRTFファイル形式でのドキュメントを生成できるため，情報の共有や商品開発のドキュメントやクライアントへの提出資料としてそのまま使える
●テストダイアログを利用することで，作成したUMLの要素に対して単体，結合，システム，受け入れ，シナリオテストについて，「説明」，「入力」，「合格基準」，「実施状況」，「結果」などを記述でき，これらをシステム全体としてまとめたレポートをHTMLやRTFで出力できる→ソフトウェアライフサイクルプロセスの管理や，XP(eXtreme Programming)に利用可能
●各種UML要素のプロパティダイアログやヘルプがきれいでわかりやすい

いて，詳細なユースケースを作成します．これには，実在する制御対象デバイスから，ユーザーの要求を満たすためのユースケースをもう一度見直してみるという意味があります．ただ単に制御デバイスと要求仕様を結びつけただけではいけません．要求仕様がどのように制御対象デバイスを通して実現されているかをもう一度見直して無駄がないか，工夫する余地がないかを分析してください．

3 ドメイン構成図を作成する

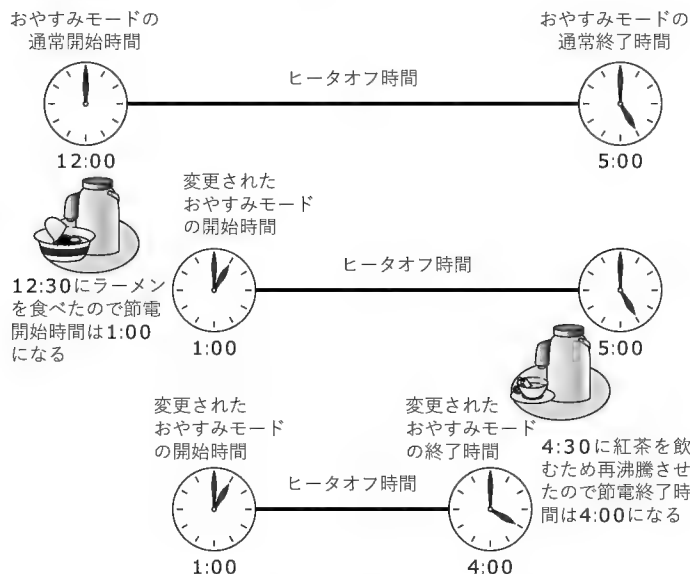
作成したユースケースを元に，ドメイン構造図を書きます．ドメイン構造図の書き方については，ユースケースを書く前に第3章でくわしく解説しましたが，UMLの表記法にしたがって分析を進めていくのであれば，ユースケースで要求分析を行ってからドメイン構造図を書き，ドメインエンジニアリングを行うのもよいでしょう．ただし体系的な再利用は，UMLやオブジェクト指向設計手法に依存しているわけではないので，第3章ではユースケースのことにふれていません．

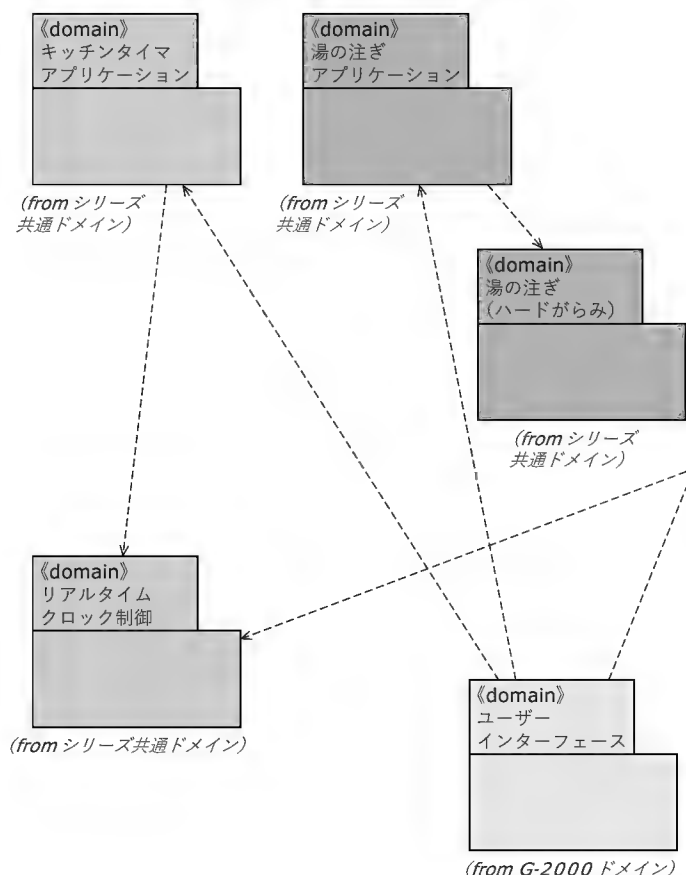
UMLツールEnterprise Architectを使った場合は，ドメイン構造図の各ドメインはパッケージとなり，Windowsのフォルダのような使い方ができます．Enterprise Architectにおけるパッケージ(フォルダ)は階層構造を取れるので「G-2000」，「G-7000」，「G-9000」，「シリーズ共通」，「上位機種共通」というパッケージを作り，そのパッケージの中にそれぞれ，「要求分析」，「分析」，「設計・実装」というパッケージを作り，「シリーズ共通」のパッケージの下の「設計・実装」のパッケージの配下に，ドメイン構造図で作成したドメイン(パッケージ)を並べていきます．このようなパッケージの構成は，Enterprise Architectを使うと簡単に行うことができ，しかも，一度構成したパッケージの階層構造を修正することもできます(**図2**)．Enterprise Architectではダイアグラムの中でパッケージの所属を，パッケージ(ドメイン)の下に“from ○○ドメイン”といった形で表記できるので，ドメインの所属を確認するのに便利です(**図3**)．

4 コア資産を実装する

ドメイン構造図で分類した商品群のコア資産について実装を行います．実装を行う過程でメンバ内のレビューによりドメインの構造を見直してドメインを統合したり，分割したほうがよいと判断した場合は，ドメインの構成を変更してもかまいません．実際にはこの変更は，Enterprise Architect上では作成したパッケージの名前を変えたり，パッケージを追加してパッケージの中に入っていたクラスを移動したりすることで実現できます．パッケージ(ドメイン)の名前の変更などは，そのパッケージ(ドメイン)を使っているダイアグラム全部に対して即座に反映されるので，図面ごとに一つ一つ直す必要はありません．これが，UMLツールを使うことのメリットの一つです．

たとえば，普及型電子ポットG-2000のドメイン構造図を書き始めた頃には「湯沸かし(ハードがらみ)」と「ヒータ制御(ハードがらみ)」は別々のドメインでしたが，後に「湯沸かし(ハードがらみ)」ドメインは「ヒータ制御(ハードがらみ)」ドメインに統合されることになります．このとき「湯沸かし(ハードがらみ)」の中にすでにクラスを作っていたのであれば，このクラスを「ヒータ制御(ハードがらみ)」のパッケージに移して，「湯沸かし(ハードがらみ)」パッケージを削除し，ドメイン構造図のドメイン間の依存関係を若干書き直すことで修正が完了します．G-2000のドメイン構造図で修正したドメイン間の依存関係は，G-7000やG-9000のドメイン構造図にも自動的に反映されます．

〔図1〕おやすみモード(節電モード)**の説明図**

〔図2〕パッケージの構成図

〔図3〕ドメインの所属の説明(一部を抜き出している)

5 具体から抽象へ

オブジェクト指向設計をベースにした開発では，分析モデルと実装モデルを分けて考え，分析モデルはより抽象度を高め，実装モデルで現実の制約条件をクリアしていくという考え方があります．たしかに，このやり方がうまくいくと分析モデルの普遍性が高まり，分析モデルの寿命が長くなるというメリットがあります．しかし，オブジェクト指向設計の初心者が抽象度の高い分析モデルを作り出すのは至難の業です．

顧客の求めるものは普遍的な一つのモデルであり，分析者の視点で見た分析モデル，設計者の視点で見た設計モデル，実装技術者の目で見た実装モデルはあくまでも視点＝ビューの違いです．モデル自体は一つなのだという考え方は頭ではわかります．しかし静的制約や動的特性で日々苦しんでいる組み込みソフトウェアエンジニアにとって「実装モデル」を頭から振り払って，最初に抽象化した「分析モデル」を考えなさいといわれても，なかなかできるものではありません．そんなときは，静的な枠組であるクラスを摘出することを考えるより，具体的にクラスの実装形態であるインスタンス(クラスを実体化したオブジェクト)を書いてしまって，要求された機能を実現できる道筋をつけてから，クラス図を書くことが有効です．このようにいくつかの「具体」を想定してからそれらの共通の特徴を抽象化していく方法は，人間の思考に近い自然な考え方です．

また，モデルを現場で効率よく運用(再利用)することを考えると，モデルは再利用や拡張方法がイメージしやすく理解しやすい適度な抽象度が望ましいと考えられます．抽象度の高い部分や高度なメカニズムに相当する部分はフレームワーク(**コラム2**参照)の中に隠ぺいし，簡単に触れられないようにしておいたほうが，モデルを使う側(アプリケーションエンジニアリング)にとって好都合です．

たとえば，自動販売機の場合を想定してみましょう．同じ自動販売機でも，切符の券売機，たばこやジュースの販売機，紙コップのコーヒーやカップラーメンの販売機などがあり，それぞれ，内部のソフトウェアは異なります．自動販売機では，おおざっぱにいうと，①金銭を投入，②品物を選択，③品物を受け取るという流れを実現するわけですが，それぞれの自動販売機で，①金銭の投入と②品物の選択の順序性が不要だったり，②品物の選択の後③品物を受け取るまで複雑な制御があったりして，扱う製品ごとに大きく内部設計が異なります．メーカーとして，すべての製品を扱うのであれば，このような自動販売機の抽象的なモデルが必要ですが，各製品用のソフトウェアを効率よく開発するには，さらに具体的にシリーズごとのソフトウェアのベースを定義し，シリーズに含まれる製品ごとのソフトウェアの変化部分をパラメータ化し，全体をフレームワーク化しておくとよいでしょう．

逆に，特定の製品のみを開発対象としているのであれば，抽象的なモデルの価値は下がり，戦略的に作られたモデルの価値が高まります．***プロダクトライン***開発でのモデルの価値は，より高い抽象度だけではなく，アプリケーションエンジニアリングにおいて，予想された期間にもっとも効率的・効果的に運用されることを考慮に入れたモデルが望まれます．ここには，トレードオフが存在します．したがって，一定期間モデルを維持し，いつか捨てる判断をするためにも，モデル化した意図や商品群のロードマップを定義しておく必要があります．

6 クラス摘出についてアドバイスするとすると……

今回のプロジェクトのコンセプトは，「商品群としての再利用性を高め，顧客に対して魅力的な商品を作ること」というものです．このように目標がはっきりしていて，魅力ある商品になるような種を仕込みつつ，多くのエンジニアにわかりやすいモデルを作るためには，比喩(メタファ)や擬人化を使うことが有効です．簡単にいえば，システムの中に小さな達人がたくさん入っていて，達人たちのスペシャリティが商品の価値を高めているという考え方をすると，多くの人にわかりやすいモデルになり

Column ②
フレームワークとは何か？

組み込みソフト開発組織にとって，新たな要求を満たすソフトウェアを，既存資産をベースにいかに効率良く開発するか？というのが長年の課題となっています．これを解決するため，さまざまな手段が考えられていますが，その中の一つに，編集する部分をできるだけ局所化し，要求の変化に対応した部分を作り込むというものがあります．極端な言い方をすれば，新規ソースコードを作らないで済ますようにするか，または開発者がソースコードにいかに触れないようにするがポイントとなります．

従来，ソフトウェアの再利用という観点でとらえられるソフトウェア部品には，よく利用するサブルーチンなどのライブラリや，IDE（Integrated Development Environment：統合開発環境）などで手軽に利用できるコンポーネントなどが一般的となっており，ソースコード規模としては比較的小さい単位となります．この場合，これらを利用する部分を開発者が作り込むという開発スタイルをとります．つまり，プログラムの入り口に近い部分〔C言語では`main()`〕から，用意されたソフトウェア部品を組み合わせたり，うまくつないだりということをして徐々にシステムを作っていきます．したがって，何もないところからシステムを構築していく場合と比較して，再利用できるソースコードがあらかじめそろっていることが，開発効率を向上させるうえでたいへん効果があります．しかし同時に，これらのライブラリやコンポーネントをどのように組み合わせるかという部分に対し，要求の変化へ対応する部分やノウハウの作り込みが集中していく傾向があるため，単なるソフトウェア部品の入れ替えによる機能拡張などへの対応が難しくなる側面も存在します．このような，積み木のようなコンポーネントでソフトウェアを構成し，いったんバラバラにするとそれぞれの部品の組み合わせがわからなくなって再構築に骨が折れるような開発を，「Building Block 型の開発」と呼びます．

一方，既存資産を再利用して組み込みソフトウェア開発を行っている場合はどうでしょうか？　開発者は，ソフトウェア構造上では（関数呼び出しツリーの）中位に位置し，処理上では中途の部分や操作画面，ドライバなどの末端に近い部分の変更を行うことが多く，ソフトウェアの全体構造および基本的な処理の流れを変更することは非常にまれです．この状況に対して見方を変えると，個々のソフトウェア部品を組み上げてソフトウェアを構成する必要はなく，むしろ再利用する単位として一つの大きな部品があって，その一部を変更して新しいソフトウェアを生み出しているととらえることができます．このように，次々に開発するソフトウェアに対してテンプレート（半完成品）を用意し，個々の開発に対応しようと考え出されたものを「フレームワーク」と呼びます．

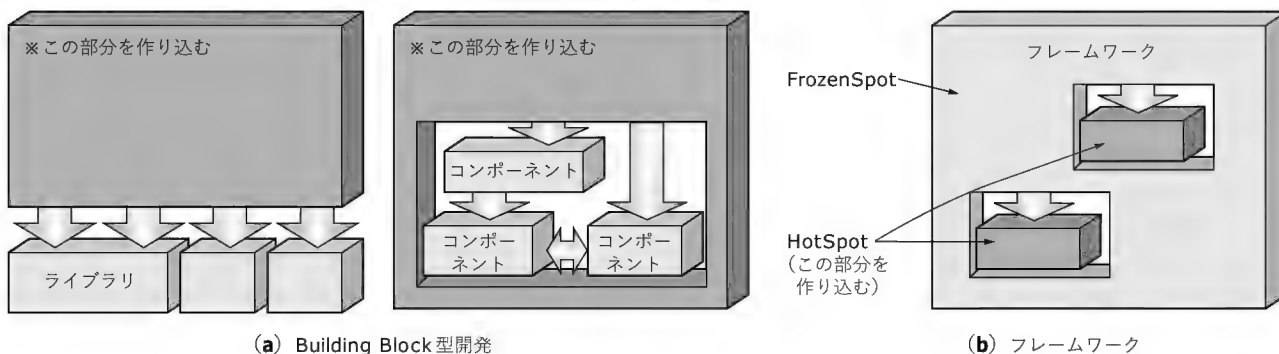
では，フレームワークの中身はどのようになっているのでしょうか？　じつはフレームワーク自体はあまり変化することのない枠組みであり，比較的大きな単位となります．また，ソフトウェアの構造や制御といった視点から，可変（変動）部分：HotSpot と不変（固定）部分：FrozenSpot に切り分けられます（**図A**）．つまり，従来のライブラリやコンポーネントを組み合わせて開発する場合と異なり，開発者は，フレームワークから利用される部分を作り込むという「逆転の発想」により，再利用部分を大きくとらえることができます．

実際の開発では，アーキテクトと呼ばれるスキルの高い技術者がドメイン分析を行うことで製品のファミリから共通部分を抽出し，個々の製品開発を横断する形でフレームワークを構築します．このとき，フレームワークの中に製品を制御する共通的・中心的なメカニズムやノウハウを不変部分として隠ぺいします．また，周辺機器へのインターフェースや製品のオプションおよび拡張機能などについては，可変部分として個別対応できるようにしておきます．

最後に，フレームワークの構築と有効に運用していくうえでの重要な点について示します．まず，フレームワークそのものについてですが，可変部分に対する仕様書を利用者に対して準備し，ノウハウを利用しやすい形で蓄積しておく必要があります．たとえば，どのようなデザインパターンを用いて実現しているかや，変更/拡張の仕方などをしっかりと記述し，利用者にとってわかりやすくしておくことがポイントです．当然のことですが，フレームワークは汎用化にすぎず，洗練されていて使いやすいことが重要です．次に，運用上のポイントですが，フレームワーク適用上の問題をアーキテクトにフィードバックし，利用者が誤解して利用することを防いだり，フレームワークの新たなバージョンを勝手に作って利用することのないよう成果物を管理していく必要があります．さらに，フレームワークにとっていつか再利用に耐えられない重大な変化が発生する時期が来ます．このようなことにタイムリに対処していくうえでも，再利用率の計測や寿命を見積もっておくことはたいへん重要です．

〔図A〕Building Block 型の開発とフレームワーク

(a) Building Block 型開発

(b) フレームワーク

ます．達人はその道のプロで，自分の仕事に対して自負があり，一般的にはユーザーにノウハウをひけらかすことはせず，よい成果だけを出すように努力しますから，オブジェクト指向の考え方にマッチしています(情報の隠ぺい，責務の明確化)．

電子ポットの場合，「お湯注ぎの達人」や「節電の達人」，「お湯沸かしの達人」，「保温の達人」などがシステムの中にいて，ドメイン分割したパッケージの中のどれかに所属し，それぞれが独自の技をもっていると考えると，クラスを作成するときにどういったクラスにすればよいかの方向性が明確になるでしょう．

たとえば「お湯注ぎの達人」は，お茶を入れるためのお湯注ぎとコーヒーを入れるためのお湯注ぎでは別々の達人がいると考えます．お茶は一定の流量でお湯を注げばよいのですが，本格的なコーヒーを入れるためにはお湯をちょろちょろと出したほうが香りを広げることができます．また，お湯注ぎだけの問題ではありませんが，お湯の温度も調節することを考えれば「新茶のための注ぎの達人」や「ウーロン茶のための注ぎの達人」なども想定できます．このような，ユーザーにとって魅力ある商品となるようなアイデアの種をちりばめて，後から安全に「達人」の追加ができるようなクラス構造にしておけば，商品の付加価値を高めるきっかけにすることができます．クラスを考えるときも，商品コンセプトを常に頭に置いておくことが重要です．

7 お湯を沸かすシナリオを考える

「お湯を沸かす」というシナリオを考えてみましょう．

- ユーザーが電子ポットのモード(沸騰，節約，ミルクのいずれか)を設定する
- 湯沸かしアプリケーションドメインで，湯の温度コントロールを行う
- 湯沸かし(ハードがらみ)で，ヒータのコントロールを行う

さて，ここで湯沸かしの温度コントロールは電子ポットの仕様書どおり，PID制御(後述)の温度テーブル方式(**表1**)で行うのがもっともシンプルであるということがわかっていますが，この部分が電子ポットのコア資産のさらに心臓部なので，5年後のことを考えるともう少し先進的な技術を取り入れる可能性を残しておいたほうが良いという気がしてきました．そこでPID制御について調べてみると，PID制御よりもファジィ制御のほうが上手にコントロールできる場合があることがわかりました．水槽に早くぴったり水を貯めるには，人間が行うような感覚で最初はジャーと水を出し，いっぱいに近づいたら徐々に蛇口を絞って最後にピタッと止めるほうが正確で早いというのです．

したがって，温度制御についても一般的なPID制御だけでなく，将来ファジィ制御で温度をコントロールできるように，湯沸かしの部分を派生させて`WaterBoilor`クラスを基底クラスにし，`PIDControlor`と`FuzzyControlor`の二つの派生クラスを派生させるような構造にしておきます．こうしておけば，さらに将来，画期的な温度制御方法が開発されても，安全に温度制御アルゴリズムを入れ替えることができます．

- **PID制御について**

PID制御は自動制御によく使われる方式で，Proportional(比例)，Integral(積分)，Derivative(微分)の頭文字をとったものです．単純なON/OFF制御で目標値に近づけると，目標値に対して行き過ぎを繰り返してしまいますが，目標値と現在値の偏差に対して制御量を「比例」させると，スムーズに目標値に近づけることができます．さらに，「積分値」を使うことでより制御を滑らかにし，「微分値」を使って素早く目標に近づくようにしたのがPID制御です．PID制御では「比例定数」，「積分定数」，「微分定数」を試行錯誤することによってチューニングしていきますが，今回はあらかじめPID制御の実験で明らかになっている偏差と制御量の関係をテーブルにしたものをポットの温度制御に使います．

〔表1〕PID制御の温度テーブル方式

		E_0(℃)				
		＜−3	≧−3	＝0	≦3	＞3
⊿ T_0(℃)	＜−3	0	100	100	100	100
	≧−3	0	70	70	70	100
	＝0	0	30	30	50	100
	≦3	0	0	0	30	100
	＞3	0	0	0	0	100

E_0：目標の水温－現在の水温
T_0：前回の制御周期時の水温－今回の制御周期時の水温

8 本当にWaterBoilorというクラス名でよいの？

`WaterBoilor`というクラスを作りましたが，本当に`WaterBoilor`とう名前でよいのでしょうか？ `WaterBoilorByPIDControl`や`WaterBoilorByFuzzyControl`というオブジェクト名はあっても何の問題もありませんが，これらのオブジェクトの静的構造で抽象概念であるクラスは`WaterBoilor`でよいのでしょうか．もしも，電子ポット商品群を作ろうとしている会社が未来永劫電子ポットしか作らない会社であることが明らかであれば，これは`WaterBoilor`でよいし，第2章で電子ポット商品群のロードマップを考えたときに電子ポット以外の商品群への再利用が話題にあがっていなかったのであればこのままでよいといえます．

しかし，もしもこの会社で電子ポット以外に「電気あんか」を作っていたり，作る計画があったりしたらどうでしょうか？このファジィ制御のクラスは電気あんかにも使えるし，他社にはない付加価値として再利用できるのではないでしょうか？そう考えると`WaterBoilor`クラスの名称は，`TemperatureControlor`のほうが良い気がします．ここでは，プロジェク

ト内のレビューで気がついたこのような修正を，せっかく考慮したファジィ制御のクラスの可能性を会社として最大限に生かすためにクラス名をTemperatureControlorとし，実装時のオブジェクトの名前をWaterBoilorにすることにします．

なお，このような「気付き」も，具体的なオブジェクトに名前を付け，クラスにするときにはもう一度抽象化できるかどうかを見直すくせをつけておけば発見しやすくなるかもしれません．ただ，ここで注意したいのは，やはりあくまで商品群のロードマップを考えたときに電気あんかの構想がまったくなかったのであれば，無理に抽象化する必要はないということです．

9 保温アプリケーションと湯沸かしアプリケーションの関係

湯沸かしが終わり保温に移行することを考えます．実装する立場でこの遷移を考えた場合，リアルタイムOSで実現するのであれば，湯沸かしと保温は一連の流れなので一つのタスクになるのではないかという問いかけが，プログラムの実装担当者からあったとします．

ドメイン分析者としては，ドメインの再利用の観点から考えて湯沸かしアプリケーションはPID制御のほかにファジィ制御で効率の良い湯沸かしを行える可能性があるし，保温のほうも今後できるだけ電力を消費しないような制御方法を新たに考える可能性もあるのでドメインを分けておきたいと考えました．

しかしよく考えると，湯沸かしの後で保温に遷移することを考えれば，今どちらの状態にいるのかを知っている人が必要なことに気がつきます．二つのドメインの状態を監視するドメインを作ることは可能ですが，そうするよりも，湯沸かしと保温は同列であるという位置づけにし，湯沸かしアプリケーションと保温アプリケーションを合体させて，湯沸かし・保温アプリケーションドメインとすることにします．この場合は，アーキテクト（実装技術者）の意見を採用し，保温アプリケーションと湯沸かしアプリケーションを合体させるという選択をしました．しかし，分析者の判断を採用して保温アプリケーションと湯沸かしアプリケーションを分離させておいてもかまいません．また，そのときの判断を後に見直したほうがよいという結論に達したならば，それは貴重な失敗体験であり，それを経験した分析者と実装技術者は次回から同じ間違いをしないでしょう．そのときのディスカッションの経過と方針がきちんと整理されていれば，どちらの方法が正しいということはありません．

大事なのは，実際にプロジェクトに参加している技術者達が納得しながら進んでいくことであり，多くの「具体」という試行錯誤を経験しながら，自分たちの守備範囲で抽象化を訓練していくことが重要です（具体から抽象へ）．最初から汎用性を意識しすぎて，無理に抽象度を高くしようとすると，具体的なオブジェクトを想像しにくくなる弊害が生まれることもあります．

10 お湯の注ぎ方の実装方法を考える

お湯の注ぎ方でコーヒーモードとお茶モードをそれぞれ注ぎ方スペシャリストがいるととらえる方法と，スペシャリストは一人で，注ぎ方レシピが二つあるととらえる方法があります．後者は具体的には，モードによってクラスを分けずに一つのクラスで，注ぎ方の方法をテーブルにして何種類かをもつという実装になります．

ソフトウェアの設計から考えると後者のほうがスマートなように思えますが，お茶注ぎとコーヒー注ぎの注ぎ方レシピは共通のフォーマットで表すことができるという前提で考えなければなりません．それよりも，お茶注ぎのスペシャリストとコーヒー注ぎのスペシャリストは別人でそれぞれが道を究めており，注ぎ方レシピも独自のものをもっていると考えたほうが，レシピのフォーマットにとらわれることなく，ユーザーの要求に応えるべくいろいろな工夫がしやすいと考えることもできます．このように判断の迷ったときは，設計者の都合よりも，少しでもユーザーの利益に近い選択肢を選ぶようにしていると，その積み重ねが顧客を満足させる商品の魅力を仕込むことにつながります．迷ったときは，商品を使ってくれる顧客の姿を思い浮かべて判断することが大事です（CSDA，Customer，Satisfaction Driven Architecture：顧客満足駆動型アーキテクチャ）．

11 おやすみモード（節電）の仕様の考察

第2章で作成した，規制によるロードマップ（架空の法律を想定）では，家電節電規制法（架空の法律）に家電業界として対応するために，電子ポットにも節電機能を仕込んでおく必要が発生しました（参照：第2章の**図4**）．

おやすみモードの仕様も，単純なタイマを使った一定時間ヒータを止める機能のほかに，操作者の使用状況を解析し，使わない時間帯だけヒータをオフにする方法や，ファジィやニューラルネットを使った学習による節電も，将来的には実装できるかもしれません．ここでは，これらの機能拡張の可能性を考えたうえで，タイマによるヒータの休止に操作者の使用状況を記憶し，休止期間を調節するという方法を採用することにします．次に，この方法を示します（**図2**も参照のこと）．

- 節電の機能
- 夜，電子ポットを使わない時間帯にヒータを停止し，ポット自体の保温性能でお湯を保温する
- 朝，ポットのお湯を沸騰させすぐに使えるようにする
- 節電機能の入り方
- おやすみボタンを押すと節電モードに入る
- おやすみモード中にもう一度おやすみボタンを押すと，節電モードが解除される

- 節電機能がオンになっているかどうかは，おやすみボタン横のLEDで判断する
- 節電モードの初期値はOFFである
- 節電の方法

① 初期の節電設定は夜12時から朝5時までとする
② 夜12時から朝5時までの間に沸騰ボタンが押されたら，次回から節電の開始・終了時間を変更する（夜12時から3時までの間に沸騰ボタンが押されたら，節電開始時間を沸騰ボタンが押された時間＋30分にする．3時から5時までの間に沸騰ボタンが押されたら節電終了時間を沸騰ボタンが押された時間＋30分にする）
③ 7日間をインターバルとし，7日の間に夜12時から朝5時まで一度も沸騰ボタンが押されなかったら，節電の開始・終了時間をデフォルト値に戻す

- 節電に関わるデータのバックアップ
- 節電に関わるデータは，バックアップ電池付きのRAM上に保持する

この機能を実現するために，`PowerSaver`の基底クラスに対して，`PowerSaverByTimeTable`クラスと`PowerSaverByFuzzy`クラスの二つのクラスを派生させ，実際には`PowerSaverByTimeTable`を実装します（**図4**）．

12 リスク分析の対策の実装

組み込みシステムにとって，リスク分析の結果と対策は再利用すべき重要な資産です．場合によっては，要求仕様を実現することよりもリスク分析の対策のほうが重要なシステムさえあるほどです．組み込みシステムの用途は明確である場合が多く，使いやすいほど人は組み込みシステムに頼ることになり，組み込みシステムは潜在的に高い信頼性・安全性が求められることになります．このような組み込み商品群に対するリスク分析の結果と対策は，商品の要求仕様が変化していくこととは対照的に不変であり，代々蓄えられながら継承されていく性質をもっています．このような組み込みシステム特有の継続的ノウハウの蓄積があるゆえに，組み込みシステムの市場に新規参入した場合，最初から信頼性の高い商品をリリースすることは難しいといえます（**コラム3**参照）．

しかし，長く市場に商品を投入し続けている企業も，このリスク分析の結果と対策を技術者の暗黙知としていたのでは，その技術者がいなくなったとたんに商品の安全性・信頼性に不安が生じてしまいます．組み込みシステムに対するリスク分析と対策は，***プロダクトライン***の活動の中でコア資産としてマネージメントされるべきです．リスク分析は，ソフトウェアの資産になる前の分析レベルでのドキュメントが重要であり，この分析結果を再利用するべきです．ここでは，リスク分析の分析結果をまとめるためのリスク分析表を紹介します（**表2**，p.112）．このリスク分析表は，もともとアメリカのFDA（Food and Drug Administration：日本の厚生労働省に相当）が輸血用の血液型を自動判別する装置に対してリスク分析した例を，普及型電子ポットG-2000に適用したものです．

- リスク分析表の書き方

リスク分析表は，想定されるリスク（障害）がどのような原因で発生するのかを分析し，そのリスクがユーザーに及ぼす危険の重要度と頻度を割り出し，リスクに対する対策を立て，対策がきちんと行われたかどうかを確認するために用います．

- 障害→操作者が被る障害の内容
- 原因→障害を引き起こす原因
- 重要度→障害の重要度

発生の可能性かまたは故障率→部品の故障率のように故障率がはっきりわかっている場合は，故障率をそのまま書き，人為的ミスのような発生の確率がはっきりしない場合には，「たまにあり」，「まれ」などといった表現で発生の可能性を記入します．

Column 3

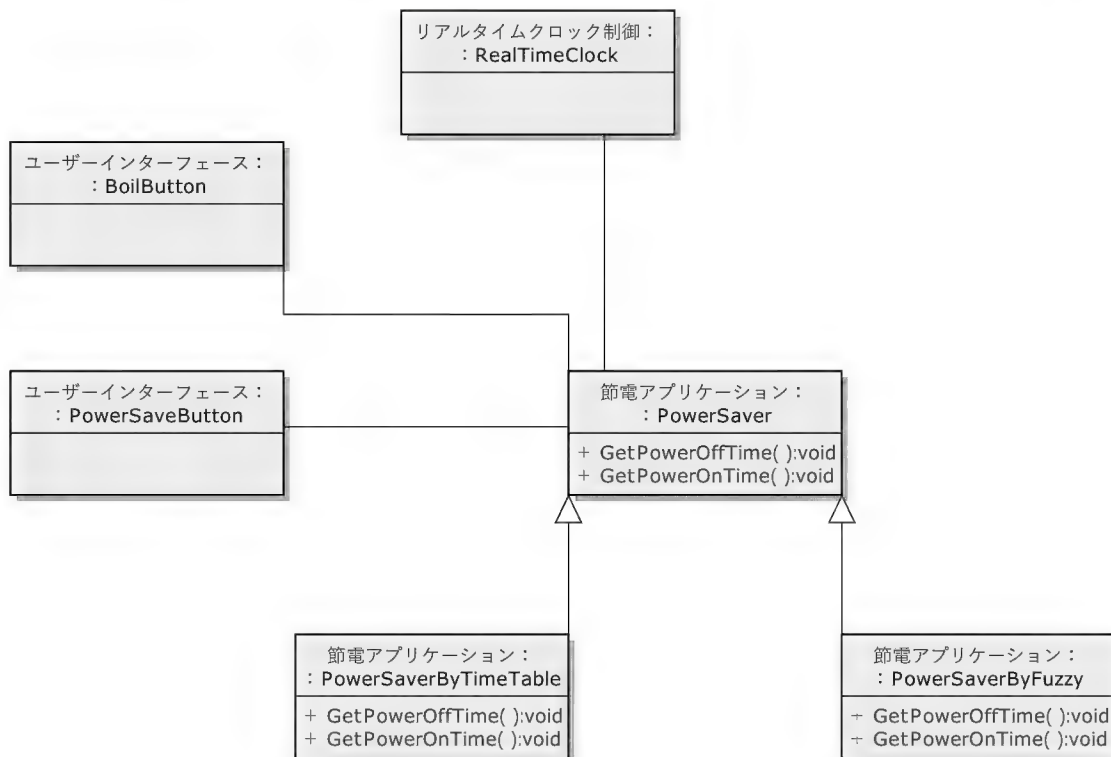
組み込みソフトウェアシステムのリスク分析について

ビジネス系のソフトウェアの開発で優れた成果を上げているエンジニアは，新たな開発対象の製品（アプリケーションソフトウェア）についての深い知識がなくとも，有効なリスク分析を行うことができるという話があります．

組み込みシステムにおいて一度も参入したことのない市場に対する商品を開発する場合，最初から有効なリスク分析を行うことは困難です．いったい，ビジネス系ソフトウェアと組み込み系ソフトウェアの違いのどこに，その差を生み出す原因があるのでしょうか？

一つは，組み込みソフトが考慮しなければならないハードウェアの誤差に，その要因があると考えられます．ビジネス系ソフトは入力系に誤差が含まれることはほとんどないですし，誤差の範囲がそれほど広くありません．たとえば，Web上でのパスワード入力で，数字とアルファベットの組み合わせで6文字以内というように入力を限定してしまえば，異常入力の範囲を限定することは容易です．一方，組み込みシステムの場合，キーを押した/押さないの1か0のポート入力を判定するだけでも，キースイッチの物理的・電気的特性によってチャタリングの除去を行う必要があり，チャタリング除去のパターンはいくらでも存在するので，エラー状態は多数想定できます．

もう一つの要因は，組み込み機器を操作する操作者のヒューマンエラーです．人間がどのような間違いを犯すかは予想が付かないし，組み込み機器に対するヒューマンエラーは無限に存在するので，組み込みソフトウェアの設計者はそれらのヒューマンエラーに優先度をつけて対処していかなければなりません．

〔図4〕おやすみモードのクラス図

対策→対策はハードウェアによる対策やソフトウェアによる対策，または，取扱説明書や注意ラベルによってユーザー自身に注意を喚起し，ユーザー自身による対策のいずれかの分類と具体的な方法を書きます．
- 実施確認の方法→設計書の番号またはテスト計画のドキュメント番号
- チェック→実施確認を行ったチェック

リスク分析の内容は，一部要求仕様書に書かれている場合もありますが，安全性や信頼性の観点から設計者以外のメンバを交えてレビューを行い，できるだけたくさんのリスク(障害)の可能性を検討し対策を講じる必要があります．また，このようにして構築したリスク分析と対策のノウハウは単一機種だけでなく，商品群として企業の製品全体に水平展開すべきコア資産であることから，データベース化することが望ましいと考えられます．データベース化されたリスク分析のノウハウには，フィールドで発生した障害に対する対策を追加したり，より安全性・信頼性が高くなるような対策への修正を行ったりしながら徐々に充実させていきます．リスク分析と対策のノウハウは

このような組み込み機器に対する操作者の無限に存在する誤用(ヒューマンエラー)に優先度をつけていくためには，過去にリリースした商品の不具合情報やクレームのデータを蓄積していることが必要です．「電子レンジに卵を入れるな」は開発当初に予測できても，濡れたペットの猫を乾かそうとして電子レンジに入れるようなユーザーがいることがわかったら，場合によっては，電子レンジ内の温度センサで動く対象物かどうかを判定し，対象物が動いていること判明したら警告メッセージを表示しなければならないかもしれません(取扱説明書に「電子レンジに猫を入れるな」と書く対策もあるが)．

組み込みシステムに限らず操作の対象が人間である場合は無限のヒューマンエラーが存在するので，これまで商品群を販売してきた経験がものをいうといえます．したがって，**表2**のようなリスク分析表は長い期間マネージメントしていくべき対象であり，組み込み商品群の貴重なコア資産です．リスク分析表をデータベースにしてマネージメントし，水平展開していくことも***プロダクトライン***の活動です．

リスク分析表についての***プロダクトライン***におけるマネージメント活動とは，商品をリリースした後にフィールドや生産工程で起こった不具合をリスク分析表にフィードバックして対策を洗練し，より効果的な対策へ変化させていくことです．このような洗練の工程は，組み込みソフトウェアエンジニアの頭の中で暗黙的に行われることが多いと思いますが，これをデータベース化して明示的に行うと，コア資産としての価値が高まります．リスク分析の対策を洗練していくためには，リスク分析の入力となる不具合や障害をデータベース化することも必要です．

※ (株)テクノホロンのWebページに，フリーのWeb対応障害管理システムが公開されています(`http://www.techno-holon.co.jp/jp/index.html`)．

〔表2〕電子ポットシリーズ共通の障害分析表の例

番　号	障　害	原　因	重要度	発生の可能性/故障率	対　策	実施確認の方法	チェック
No.	Hazard	Cause	Level of Concern	Likelihood/ Failure Rate	Method of Control	Trace	Check
A-1	ヒータの異常加熱により火災が起こる	●サーミスタの故障 ●ヒータの故障	High	1/10000 (故障率)	**●ハードウェアによる対策** 温度ヒューズによるヒータへの回路切断 **●ソフトウェアによる対策** ブザーによる注意喚起とエラー表示(30秒)を行う	設計書番号 #001 テスト計画 #001	□ □
		●水の量が少ないのに加熱した	High	たまにあり (Mederate)	**●ソフトウェアによる対策** 第1水位センサがオフ状態ならば，ヒータや沸騰ボタンは動作しない	設計書番号 #002 テスト計画 #002	□ □
A-2	ポット内の水温が上がらないのでお湯が使えない	●ヒータが動作しない ●ヒータの動作が不安定	Low	まれ(Low)	**●ソフトウェアによる対策** ヒータ制御中に1分周期で水温を検出し，目標温度よりも水温が5℃下がり，かつ前回検出した水温よりも今回検出した水温の方が低い場合，ヒータ電源をオフにしブザーによる注意喚起とエラー表示(30秒)を行う	設計書番号 #003 テスト計画 #003	□ □
A-3	お湯で火傷する	●蓋が開いたのにヒータがオンのままになっている	High	まれ(Low)	**●ソフトウェアによる対策** 蓋センサがアクティブになったら，ヒータを停止し，沸騰ボタンを効かないようにする． ブザーによる注意喚起とエラー表示(30秒)を行う	設計書番号 #004 テスト計画 #004	□ □
		●お湯を出すつもりがないのに誤って給湯ボタンを押してしまった	High	たまにあり (Mederate)	**●ソフトウェアによる対策** 給湯はロック解除ボタンを押さないと効かないようにする． 給湯してから5分たったら給湯ボタンをロックする	設計書番号 #005 テスト計画 #005	□ □
		●子どもがいたずらし，給湯してしまった	High	まれ(Low)	**●ソフトウェアによる対策** 給湯はロック解除ボタンを押さないと効かないようにする． 給湯してから5分たったら給湯ボタンをロックする **●ユーザー自身による対策** 「子どもの手の届かないところにポットを設置する」ように取り扱い説明書に記載し注意を促す	設計書番号 #006 テスト計画 #006	□ □
A-4	蒸気で火傷する	●ポットの蓋を開けるときに蒸気で火傷する	High	たまにあり (Mederate)	**●ユーザー自身による対策** 「上ぶたを開けるときは，蒸気にご注意ください」のラベルを上ふた付近に貼って注意を促す	設計書番号 #007 テスト計画 #007	□ □
A-5	ポットの中で雑菌が繁殖しお腹をこわす	●保温設定で60℃を選択したため耐熱性の雑菌が繁殖した	High	まれ(Low)	**●ソフトウェアによる対策** 保温モードで90℃と60℃を選んだときも一度沸騰させてから目標温度まで冷やす	設計書番号 #008 テスト計画 #008	□ □

出典：Guidance for FDA Reviewers – Premarket Notification Submissions for Automated Testing Instruments Used in Blood Establishments –
`http://www.fda.gov/cber/gdlns/pmaautotest.pdf`

それ自体がソフトウェア資産ではないですが，これも商品群における***プロダクトライン***の活動によるコア資産のマネージメントにあたります．

実際に分析した障害の対策は，基本的には障害の監視ドメインによって，電子ポットの機能とは独立してウォッチするバックグラウンドの監視と，ユーザーインターフェースドメインや各アプリケーションドメインにおける異常入力の監視の二つによって実施されます．

Column ④

ソフトウェアのVerification(検証)とValidation(妥当性の確認)について

General Principles of Software Validation; Final Guidance for Industry and FDA Staff 3.1.2 Verification and Validation から，一部修正してとりあげます．

多くのソフトウェア技術ジャーナルの記事の中で，Verification(検証)とValidation(妥当性の確認)の用語を交換可能なものとして扱ったり，Verification(検証)とValidation(妥当性の確認)Test(テスト)：(VV&T)の三つの用語を，まったく差異のない一つの概念であるかのように言及している場合があります．

ソフトウェア検証は，ソフトウェア開発のライフサイクルのある段階での設計のアウトプットが，その段階における明確な要求事項にすべて適合しているということの客観的な証拠を提供します．ソフトウェア検証では，ソフトウェアが開発される過程における一貫性，完全性，正確性が証明されることと，また，それを補間する文書が存在することが期待されています．そして，その積み重ねの結果がValidation(妥当性の確認)されているという結論に結びつきます．ソフトウェアテストは，ソフトウェア開発のアウトプットが，そのインプット要求事項に適合することを確実にするための検証活動の一つです．他の検証活動には，さまざまな統計的，動的分析，ソースコードや文書の検査，ウォークスルー，レビューなどが含まれます．

ソフトウェアの妥当性確認は，「ソフトウェアの仕様がユーザーニーズおよび意図された用途に適合していること，そしてソフトウェアの開発を通して実施される要求事項の確認が客観的な証拠によって提供されること」と定義することができます．実際には，ソフトウェアの妥当性確認の活動は，ソフトウェア開発ライフサイクルの途中，そして終了時には，すべての要求事項が満足されているのを確実にするために実施されることになります．

ソフトウェアの妥当性確認は，すべての要求事項が正しく完全に実施され，そしてシステムへの要求事項が満足されているのを示す証拠をトレースすることが可能になっている必要もあります．ソフトウェアの妥当性が確認されているという結論は，ソフトウェア開発ライフサイクルの各段階において実施される包括的なソフトウェアテスト，検査，分析，そしてその他の検証活動に大きく依存しています．ソフトウェアは通常，ハードウェアを含むシステム全体の一部ですから，シミュレーションによる製品の動作環境やユーザーサイドで行われる製品のソフトウェア機能のテストは通常，製品全体の設計バリデーションの中に含まれます．

具体的な例をあげると，たとえば，あるカーナビゲーションシステムのアプリケーションソフトをヨーロッパのソフト会社で作らせ，日本向けにパッケージ化し日本で発売したとします．日本での発売後，アプリケーションの中で表示される日付が“日，月，年”の並びであることがわかりました．このソフトの設計仕様書はヨーロッパのソフト会社に作らせていたため，この日付の表現は設計仕様書どおりでした．この例では，設計仕様に基づいてテストを行い検証はパスしたものの，妥当性の確認が十分に行われていなかったために，ユーザーの要求を完全に満たすことができなかったということになります．検証はOKでも，妥当性の確認はNGというのはあり得るということです．

*　　　*

開発者は，ソフトウェアの品質を高めるための製品に対するテストを永遠に続けることはできません．というよりは，十分にテストを行う間もなく製品を発売する期限が刻一刻と迫ってくるというのが現実でしょう．そう考えると，ソフトウェアの妥当性確認はどこまで，また，いつまでやればよいのでしょうか？　現実を考えると，求められているユーザー要求に製品が適合しているという「自信」が十分なレベルに達するまで，ソフトウェアValidationは行う必要があると考えられます．仕様書の中で見つけられた間違いの修正数や，残された不具合の評価，テストカバレッジの結果などは，製品を出荷してよいかどうかの確信のレベルを得るために使われます．また，自信のレベルと必要とされるソフトウェアValidation，検証，テスト作業の程度は，製品の安全上のリスク(ハザード)に起因すべきです．想定したリスク(ハザード)に対して漏れのないようにValidationを行う必要があり，また，リスク(ハザード)がユーザーに与える影響が大きければ大きいほどValidationは慎重かつ確実に行われる必要があります．

13 保守用ソフトウェアの設計と実装

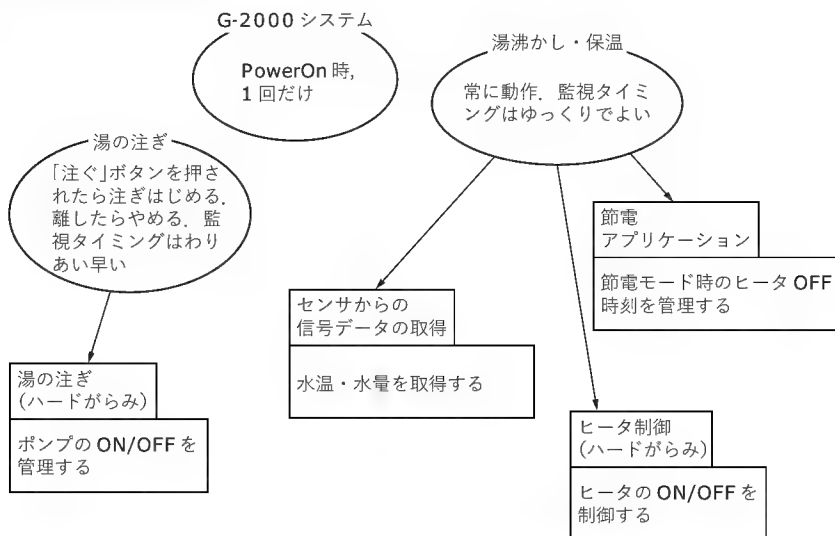
保守用のソフトウェアは，製品の要求仕様に比べると後回しにされがちですが，製品を生産する際，また，出荷前の調整，出荷した後に保守時に必要になるので，ほとんどの組み込みソフトウェアシステムに実装されます．保守用のソフトウェアについても，できるだけ開発の初期段階，要求分析のフェーズで必要とされる機能を洗い出しておくべきでしょう．Appendix **図5**に，G-2000の保守用のユースケースをドメイン構造図にマッピングした図を示します．保守要求は通常，商品に搭載されている各種のデバイスに対して行われるため，サブアクタとして摘出したデバイスを検査するという形で保守の要求仕様を実現することになります．

14 コア資産の実装

システムはすでに分析者によりドメインに分割されていて，コア資産として再利用していくドメインは選定されています．このコアとなるドメインを，実際のソースコードに落としていきます．**表3**に示すようなドメインについて，ソースコードを作成しました．

● 作成したソースコードについて

本特集では，作成したソースコードをMicrosoft Visual C++

〔図5〕タスク分割

でシミュレーションできるようにしています．ソースコード全体の説明を**表3**に示しました．また，誌面に掲載されたソースコードと掲載しきれなかったソースコード，およびMicrosoft Visual C++のプロジェクトファイルを本誌のダウンロードページ(`http://www.cqpub.co.jp/interface/download/contents.htm`)からダウンロードできるようにする予定です．

15 タスクをどのように分割すべきか？

UMLでクラス図，コラボレーション図を作成したことにより，ドメイン内で実現すべき

〔表3〕ソースコードの説明

再利用性	ドメイン	ファイル名	ファイルの説明
実機でも使えるがシステムにより変更が必要	G-2000システム	GomaPot.cpp	システム全体の取りまとめ
		GomaPot.h	
実機でもそのまま使えるアプリケーション	湯沸かし・保温アプリケーション	Typedef.h	共通のデータ型定義
		TemperatureMonitor.cpp	温度監視を行う
		TemperatureMonitor.h	
		TemperatureController.cpp	温度制御のインターフェースを決める基本クラス
		TemperatureController.h	
		OnOffController.cpp	ON-OFF方式の温度制御
		OnOffController.h	
		PIDController.cpp	PID方式の温度制御
		PIDController.h	
		PIDControlTable.cpp	PID方式の温度制御が使用する制御量のテーブル
		PIDControlTable.h	
	湯の注ぎアプリケーション	PouringController.cpp	お湯注ぎを行う
		PouringController.h	
		PouringMaster.h	お湯注ぎのインターフェースを決める基本クラス
		PouringForTea.h	お茶用のお湯注ぎ
		PouringForCoffee.cpp	コーヒー用のお湯注ぎ
		PouringForCoffee.h	
		PourCoffeeTable.cpp	コーヒー用のお湯注ぎが使用する制御量のテーブル
		PourCoffeeTable.h	
	節電アプリケーション	PowerSaver.cpp	節電の基本クラス
		PowerSaver.h	
		PowerSaverByTimeTable.cpp	タイムテーブルを使用する節電
		PowerSaverByTimeTable.h	
ハードにアクセスするため本番用は書き換えが必要	ヒーター制御(ハードがらみ)	HeaterController.cpp	ヒーターの制御
		HeaterController.h	
	センサからの信号データの取得	SensorInfo.cpp	センサの情報を取得する
		SensorInfo.h	
	湯の注ぎ(ハードがらみ)	MotorController.h	ポンプの制御
	リアルタイムクロック制御	RealTimeClock.h	時刻の取得
テスト用なので実機では使用できない		Water.h	ヒーター制御による水温の変化をシミュレーションする
		Water.cpp	
		Cup.cpp	お湯を注いだ時のカップに注がれる水量をシミュレーションする
		Cup.h	
		Rtos.h	リアルタイムOSの関数をシミュレーションする
		Rtos.cpp	
		UserInterface.h	ユーザーインターフェース

機能分割はすでにできあがっています．次に，これらの機能を実際にCPU上で動かすためには，時間的な分割を行う必要があります．今回は電子ポット商品群に対してμITRON仕様のリアルタイムOSを使用することにしたので，疑似平行動作を行うためにタスク（スレッド）をどのように分割すべきかを考えます．

UMLで分析したクラスオブジェクトの単位がタスクの単位とある程度一致していればよいのですが，機能的モジュール分割と時間的モジュール分割は必ずしも一致しないということを考慮し，アーキテクト（実装者）は設計を進めなければなりません．リアルタイムOSを使った時間的モジュール分割は，ある程度の規模をもった組み込みソフトウェアでは必要不可欠な技術ですから，組み込みソフトの実装経験があるアーキテクトにとっては腕の見せどころでもあります．

それではまず，どんなタスクが必要になるのか考えるために，大まかな機能や起動タイミングなどをドメインごとに考えてみます．

- **G-2000システム**：各ドメインの生成，関連付けを行います．起動時に1回だけ実行するので，タスクとはせずOSの初期化前に実行します．
- **湯の注ぎアプリケーション**：「給湯」ボタンが押されたらお湯を注ぎ，離すと注ぐのを停止します．ボタンを押されるのを待っているか，常にボタンを監視してボタンの状態に合わせて動作する方法が考えられますが，タスクが必要と思われます．
- **湯の注ぎ（ハードがらみ）**：ここはハードウェアにアクセスする部分です．湯の注ぎアプリケーションがお湯を出せといってきたら出し，止めろといわれたら止める，主体的には動かないパッシブなドメインです．なのでタスクは必要ないと思われます．
- **湯沸かし・保温アプリケーション**：ここは，電源が入っている間は常に水温を監視し一定に保つ機能と，蓋を閉じたり沸騰ボタン押下時は水を沸かす機能があります．水温はそれほど急激に変化するものではないので，極端に短い間隔で監視する必要はありませんが，タスクが必要です．
- **節電アプリケーション**：節電アプリケーションは，時刻を監視し使用される時間を予測します．長い時間使用されない間はヒータの電源を切り，使用時には適温になるように自動的に調整します．湯沸かし保温アプリケーションから実行されるパッシブなオブジェクトとするか，節電アプリケーションがアクティブに動いて湯沸かし保温アプリケーションに通知するかの

Column

機能的分割と時間的分割について

ドメインの分割やクラスの摘出は，基本的にはシステムの機能を扱いやすい単位に分割し整理していく作業です．ただし，組み込みシステムにおいて静的制約や動的特性の重要度が高い場合は，これらの条件を考慮せずに抽象度の高いモデリングをめざすわけにはいきません．なぜなら，抽象度の高いモデリングを行った場合，クラスオブジェクトを生成する際のコスト（CPUパフォーマンスやROM，RAMの容量）が高くつき，結果的に実装できないという事態を生じることがあるからです．このようなオブジェクト生成のコストを軽減するために，優れたアーキテクチャの技術を導入したり，そのようなアーキテクチャ自体を再利用したりすることはありますが，一般的には静的制約や動的特性の重要度が高い場合は，抽象度の高いモデルを作ることよりも，実装の実現を重視することが多いと考えられます．

しかし，組み込みシステムによっては要求の多様化を重視する必要性から高性能のCPUを導入するなどして静的制約を軽減することもあります．このような場合は，システムのモデルの抽象度を高くするために動的特性に関連の深い時間的分割よりも機能的分割に重点が置かれることになります．要求の多様化に起因するモデルの抽象度向上と静的制約や動的特性とのトレードオフは，本特集第1章で論じた組み込みシステムの特徴分析の結果をもとによく考えなければならない問題です．

要求の多様化の必要性が低い場合，静的制約や動的特性をとくに重視して組み込みソフトウェアシステムにリアルタイムOSを導入し，設計当初からシステムをタスク（スレッド）に分割していくという方法を取ることがあります．この場合のモジュールの分割は時間的分割に重点を置くことになります．

リアルタイムOSを使う場合，一つ一つのタスクが優先度をつけて平行に動くことができるので，タスク分割は時間的な分割単位としての性格が強いといえます．リアルタイムOSにおけるタスクは機能分割の単位とも考えられますが，時間的分割と機能的分割の結合度が強いと見たほうがよいでしょう．

しかし，このような時間的分割に根ざした設計を採用するのは静的制約や動的特性を重視しているからであって，要求の多様化や機能モジュールの体系的な再利用を考慮する場合は，機能的分割と時間的分割のバランスをよく考える必要があります．

電子ポット商品群のコア資産の設計・実装ではまず，機能分割を先に考えてドメインエンジニアリングを行い，クラスをオブジェクトとして実装するフェーズにおいて時間的な分割（オブジェクトやメソッドのタスクへのマッピング）を実装技術者（アーキテクト）に考えてもらうという手順を踏みました．これは，***プロダクトライン***の体系的再利用を考慮し，電子ポットの静的制約や動的特性を認識しつつ，要求の多様化とのバランスを考えた結果です．

このように，機能的分割から時間的分割への流れ，機能分割したクラスオブジェクトを時間的分割の単位であるタスク（スレッド）へマッピングする際の設計方針は，組み込みソフトウェアシステムの特徴である要求の多様化，静的制約，動的特性，生産性の向上のバランスによっては変わる可能性があり，決して一意に決まるものではないということを，組み込みシステムの分析者や実装技術者は，設計を開始する前によく考えておくことが必要です．

〔リスト1〕TemperatureController.h

```
////////////////////////////////////////////////////////
//  ファイルの説明：湯沸かし制御クラスヘッダ
////////////////////////////////////////////////////////
#ifndef __TEMPERATURE_CONTROLLER_H
#define __TEMPERATURE_CONTROLLER_H


#include "Rtos.h"
#include "Typedef.h"
#include "SensorInfo.h"
#include "HeaterController.h"

////////////////////////////////////////////////////////
//  ドメイン    ：湯沸かしアプリケーション
//  クラスの説明：湯沸かし制御クラス
////////////////////////////////////////////////////////
class TemperatureController
{
//////////////////////////////////
//  データメンバ
//////////////////////////////////
protected:
    SHORT     controlPeriod;                         // 制御間隔
    SHORT     controlRatio;                          // 操作量
    SHORT     targetTemperature;                     // 目標温度

    SensorInfo          * pSensorInfo;
    HeaterController    * pHeater;
    ID        waitTaskID;    // WaitControlPeriod()実行中のタスクID
////////////////////////////////////////////////////////
//  メソッド
////////////////////////////////////////////////////////
public:
    TemperatureController( );
    // 制御周期取得
    SHORT getControlPeriod( );
    // 制御周期設定
    VOID setControlPeriod( SHORT ControlPeriod );
    // 操作量設定
    VOID setControlRatio( SHORT ControlRatio );
    // 操作量と制御周期によりヒータを制御する
    VOID waitControlPeriod();
    // ヒータ制御の中断
    VOID breakWaitControlPeriod();
    // 目標温度設定
    VOID setTargetTemperature( SHORT TargetTemperature );
    // 湯沸かし制御の開始前の初期化
    virtual VOID initControl() = 0;
    // 温度制御
    virtual SHORT control() = 0;

    // センサ情報取得クラスの設定
    VOID setSensorInfo( SensorInfo * pSensorInfo );
    // ヒータクラスの設定
    VOID setHeater( HeaterController * pHeater );

};

#endif // __TEMPERATURE_CONTROLLER_H
```

〔リスト2〕TemperatureMonitor.h

```
////////////////////////////////////////////////////////
//  ファイルの説明：温度監視クラスヘッダ
////////////////////////////////////////////////////////
#ifndef __TEMPERATURE_MONITOR_H
#define __TEMPERATURE_MONITOR_H

#include "Rtos.h"
#include "TemperatureController.h"
#include "PIDController.h"
#include "OnOffController.h"
#include "PowerSaver.h"
#include "TemperatureMonitor.h"
#include "UserInterface.h"

////////////////////////////////////////////////////////
//  ドメイン    ：湯沸かしアプリケーション
//  クラスの説明：温度監視クラス
////////////////////////////////////////////////////////
class TemperatureMonitor
{


////////////////////////////////////////////////////////
//  定数定義
////////////////////////////////////////////////////////
#define      MONITOR_CONTROL_PERIOD       (60)      // 制御周期
#define      STERILIZE_TEMP           (100)         // 殺菌温度
#define      STERILIZE_TIME           (3*60)        // 殺菌時間

////////////////////////////////////////////////////////
//  データメンバ
////////////////////////////////////////////////////////
protected:
    enum e_temp_cond
    {
         TEMP_COND_KEEP = 0,                //保温状態
         TEMP_COND_BOIL,                    //沸かしている状態
         TEMP_COND_STERILIZE,               //殺菌状態
    };
    e_temp_cond    tempertureCondition;    //保温，沸かす状態

    TemperatureController    * boilControl;
                                            // 湯沸かしオブジェクト
    TemperatureController    * sterilizeControl;
                                  // 湯沸かし後の殺菌オブジェクト
    TemperatureController    * keepControl; // 保温オブジェクト
    TemperatureSelector * temperatureSelectorObj;
                                            // 温度選択オブジェクト
    BOOL                     reboilFlag;    // 沸かし直しフラグ
    PowerSaver               * powerSaverObj; // 節電オブジェクト
    HeaterController         * heaterControllObj;// ヒータ制御
////////////////////////////////////////////////////////
//  メソッド
////////////////////////////////////////////////////////
public:
    TemperatureMonitor();
    //温度監視タスク
    VOID monitorTemperatureTask( INT StaCd );
    //湯沸かし方式の設定
    VOID setBoilControl( TemperatureController * pBoil )
    {
        this->boilControl = pBoil;
    };
    //殺菌方式の設定
    VOID setSterilizeControl( TemperatureController * pSterilize )
    {
        this->sterilizeControl = pSterilize;
    };
    //保温方式の設定
    VOID setKeepControl( TemperatureController * pKeep )
    {
        this->keepControl = pKeep;
    };
    //温度選択の設定
    VOID setTemperatureSelector( TemperatureSelector * tempSelect )
    {
        this->temperatureSelectorObj = tempSelect;
    };
    // 節電オブジェクトの設定
    VOID setPowerSaver( PowerSaver * powerSaverObj)
    {
        this->powerSaverObj = powerSaverObj;
    };
    // ヒータ制御オブジェクトの設定
    VOID setHeaterController( HeaterController * heaterControllObj)
    {
        this->heaterControllObj = heaterControllObj;
    };
    //沸かし直し
    VOID reboil(  );

protected:
    //目標温度まで加熱する
    VOID boil( SHORT Target );
    //殺菌
    VOID sterilize( SHORT Time, SHORT Target );
    //保温
    VOID keep( );
};

#endif   // __TEMPERATURE_MONITOR_H
```

〔リスト3〕GomaPot.h

```
/////////////////////////////////////////////////////////
//  ファイルの説明：GOMA ポットのシステムクラスヘッダ
/////////////////////////////////////////////////////////
#ifndef __GOMA_POT_H
#define __GOMA_POT_H

#include "Typedef.h"
#include "TemperatureMonitor.h"
#include "SensorInfo.h"
#include "HeaterController.h"
#include "TemperatureController.h"
#include "PIDController.h"
#include "OnOffController.h"
#include "PouringController.h"
#include "PouringForTea.h"
#include "PouringForCoffee.h"
#include "PowerSaverByTimeTable.h"
#include "RealTimeClock.h"
#include "UserInterface.h"

/////////////////////////////////////////////////////////
//  ドメイン    ：G-2000 システム
//  クラスの説明：GOMA ポットのシステム
/////////////////////////////////////////////////////////
class GomaPot
{
protected:

/////////////////////////////////////////////////////////
//  定数定義
/////////////////////////////////////////////////////////
#define  CONTROL_PERIOD (60)           // 湯沸しの制御単位時間

/////////////////////////////////////////////////////////
//  データメンバ
/////////////////////////////////////////////////////////
protected:
    //注ぐ
    MotorController     motorControllerObj; // モータ制御
    PouringController   pouringControllerObj;   // お湯注ぎ制御
    PouringForCoffee    pourForCoffeeObj;   // コーヒーを注ぐ
    PouringForTea       pourForTeaObj;      // お茶を注ぐ

    //温度監視
    TemperatureMonitor monitorTemperatureObj;  // 温度監視
    SensorInfo              sensorInfoObj;      // センサ情報
    HeaterController        heaterObj;          // ヒータ制御
    PIDController           PIDControlObj;      // PID 温度制御
    OnOffController         onOffControlObj;    // ON-OFF 温度制御

    //節電モード
    PowerSaverByTimeTable   powerSaverObj;      // 節電モード

    //ユーザーインターフェース
    TemperatureSelectortempSelect;                        // 設定温度取得
    PouringSelector         pouringSelectorObj;       // 注ぎかた管理
    PourButton              pourButtonObj;            // 注ぐボタン
    BoilButton              boilButtonObj;            // 沸騰ボタン
    PowerSaveButton         powerSaveButtonObj;       // 節電ボタン

    RealTimeClock       realTimeClockObj;  // リアルタイムクロック

/////////////////////////////////////////////////////////
//  メソッド
/////////////////////////////////////////////////////////
public:
    //生成
    VOID create();
    //温度監視タスク
    VOID controlTempTask( INT StaCd )
    {
        this->monitorTemperatureObj.monitorTemperatureTask( StaCd );
    }
    //お湯注ぎタスク
    VOID pouringTask( INT StaCd )
    {
        this->pouringControllerObj.pouringTask( StaCd );
    }
};

/////////////////////////////////////////////////////////
//  グローバル関数
/////////////////////////////////////////////////////////
//温度監視タスク
VOID     monitorTempMain( INT StaCd );
// システムオブジェクト取得
GomaPot * getGomaObj(  );

#endif // __GOMA_POT_H
```

いずれかの方法が考えられます．湯沸かし保温アプリケーションが常に動作していることを考えると，同じような周期で動作する節電アプリケーションをあえてタスクとする必要もなさそうなので，今回はパッシブなオブジェクトとします．

○ **センサからの信号データの取得**：水量，水温などセンサからの情報を取得します．ここもパッシブなオブジェクトで，タスクは必要ないと思われます．

○ **ヒータ制御(ハードがらみ)**：ヒータのON/OFFを行いますが，他のオブジェクトから指示されて受動的に動くのでタスクは必要ありません．

*　　　　*

以上の考え方で，アクティブなオブジェクトとパッシブなオブジェクトの関係を表したのが**図5**です．G-2000システムは起動時に1回だけ起動されます．他のほとんどのオブジェクトを保有していて，それらの関連付けを行います．

湯の注ぎアプリケーションと湯沸かし・保温アプリケーションは，それぞれタスクとして平行に動くものとして考えます．関連もまったくありません．湯の注ぎ(ハードがらみ)は，湯の注ぎアプリケーションが使用します．節電アプリケーション，センサからの信号データの取得，ヒータ制御(ハードがらみ)は，湯沸かし，保温アプリケーションに使用されます．

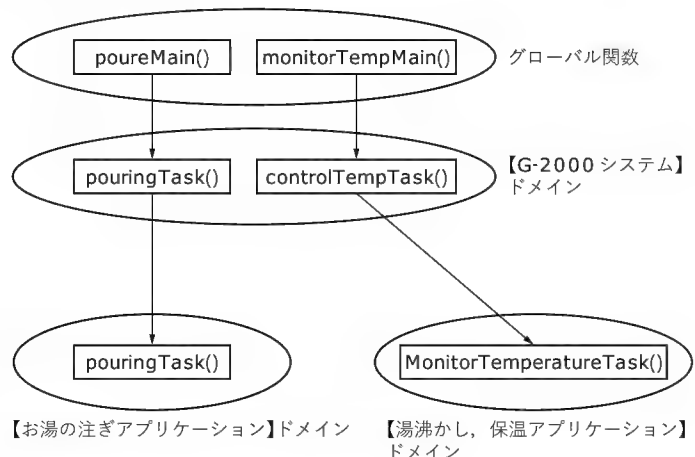
16 G-2000システムの実装

G-2000システムドメイン内で，小さなオブジェクトを除いたほとんどのオブジェクトを生成しています．オブジェクト間の関連もここで把握していて，それらの関連付けを行います．

たとえば，湯沸かし，保温アプリケーションは，状況に応じてON/OFF制御方式とPID制御方式を使い分けます．どの状況でどの制御方式を使用するのかをここで設定します．お湯を沸かすときと殺菌するときはON/OFF制御方式，保温するときはPID制御方式を設定しています．それぞれのクラスは`TemperatureController`(`TemperatureController.h`：**リスト1**)クラスから派生しているので，実際にお湯を沸かす`TemperatureMonitor`(`TemperatureMonitor.h`：**リスト2**)クラスは，その基本クラスのポインタ型として各クラスオブジェクトを受け取ります．かりにこのクラスを入れ替えたり将来`FuzzyController`クラスが開発さ

〔リスト4〕GomaPot.cpp

```
///////////////////////////////////////////////////////////
//  ファイルの説明：GOMA ポットのシステムクラス関数
///////////////////////////////////////////////////////////
#include "Rtos.h"
#include "GomaPot.h"


///////////////////////////////////////////////////////////
//  クラス：GOMA ポットシステム
//  機　能：GOMA ポットシステムの生成
//              使用するクラスの初期化を行う
//  引き数：なし
//  戻り値：なし
///////////////////////////////////////////////////////////
VOID GomaPot::create()
{

    // ControlTemperature（湯沸かし）クラスの初期化
    this->PIDControlObj.setControlPeriod( CONTROL_PERIOD );                         // 制御周期設定
    this->PIDControlObj.setSensorInfo( &(this->sensorInfoObj) );
    this->PIDControlObj.setHeater( &(this->heaterObj) );
    this->onOffControlObj.setControlPeriod( CONTROL_PERIOD );                       // 制御周期設定
    this->onOffControlObj.setSensorInfo( &(this->sensorInfoObj) );
    this->onOffControlObj.setHeater( &(this->heaterObj) );

    //TemperatureMonitor（温度監視）の初期化
//  this->monitorTemperatureObj.setBoilControl( &(this->PIDControlObj) );           // 湯沸かし方式設定
    this->monitorTemperatureObj.setBoilControl( &(this->onOffControlObj) );         // 湯沸かし方式設定
    this->monitorTemperatureObj.setSterilizeControl( &(this->onOffControlObj) );    // 殺菌方式の設定
    this->monitorTemperatureObj.setKeepControl( &(this->PIDControlObj) );           // 保温方式の設定
    this->monitorTemperatureObj.setTemperatureSelector( &tempSelect );              // 温度選択クラス
    this->monitorTemperatureObj.setPowerSaver( &(this->powerSaverObj) );            // 節電オブジェクトの設定
    this->monitorTemperatureObj.setHeaterController( &(this->heaterObj) );          // ヒータ制御オブジェクトの設定

    // pouringController（注ぎ制御）の初期化
    this->pourForCoffeeObj.setMotorControllerObj( &(this->motorControllerObj) );    // モータ制御の指定
    this->pourForTeaObj.setMotorControllerObj( &(this->motorControllerObj) );       // モータ制御の指定
    this->pouringControllerObj.setMasterOfCoffee( &(this->pourForCoffeeObj) );     // コーヒーオブジェクト設定
    this->pouringControllerObj.setMasterOfTea( &(this->pourForTeaObj) );            // お茶オブジェクト設定
    this->pouringControllerObj.setPouringSelector( &(this->pouringSelectorObj) );   // 注ぎかた選択オブジェクト設定
    this->pouringControllerObj.setPourButton( &(this->pourButtonObj) );             // 注ぎボタンオブジェクト設定

    //節電アプリケーションの初期化
    this->powerSaverObj.setRealTimeClockObj( &(this->realTimeClockObj) );          // リアルタイムクロック設定

}

///////////////////////////////////////////////////////////
//  グローバル関数
///////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////
//  機　能：温度監視タスク
//  引き数：未使用
//  戻り値：なし
///////////////////////////////////////////////////////////
VOID monitorTempMain( INT StaCd )
{
    GomaPot * gomaObject = ::getGomaObj(  );
    gomaObject->controlTempTask( StaCd );
}

///////////////////////////////////////////////////////////
//  機　能：お湯注ぎタスク
//  引き数：未使用
//  戻り値：なし
///////////////////////////////////////////////////////////
VOID poureMain( INT StaCd )
{
    GomaPot * gomaObject = ::getGomaObj(  );
    gomaObject->pouringTask( StaCd );
}

///////////////////////////////////////////////////////////
//  機　能：システムオブジェクト取得
//  引き数：なし
//  戻り値：システムオブジェクトのポインタ
///////////////////////////////////////////////////////////
GomaPot * getGomaObj(  )
{
    static GomaPot gomaPotObject;
    return &gomaPotObject;
}
```

〔図6〕タスクの入り口

〔リスト5〕PouringController.h

```
////////////////////////////////////////////////////////
//  ファイルの説明：お湯注ぎクラスヘッダ
////////////////////////////////////////////////////////
#ifndef __PURING_CONTROLLER_H
#define __PURING_CONTROLLER_H

#include "Typedef.h"
#include "UserInterface.h"
#include "PouringMaster.h"


////////////////////////////////////////////////////////
//  ドメイン    ：湯の注ぎアプリケーション
//  クラスの説明：お湯注ぎクラス
////////////////////////////////////////////////////////
class PouringController
{
////////////////////////////////////////////////////////
//  データメンバ
////////////////////////////////////////////////////////
protected:
    PouringMaster * pMasterOfCoffee;       //コーヒー専用
    PouringMaster * pMasterOfTea;          //お茶専用
    PouringSelector * pPouringSelector;    //注ぎ方法選択
    PourButton * pPourButton;              //注ぎボタン

////////////////////////////////////////////////////////
//  メソッド
////////////////////////////////////////////////////////
public:
    //コーヒーオブジェクト設定
    VOID setMasterOfCoffee( PouringMaster * CoffeeMaster )
    {
        this->pMasterOfCoffee = CoffeeMaster;
    };
    //お茶オブジェクト設定
    VOID setMasterOfTea( PouringMaster * TeaMaster )
    {
        this->pMasterOfTea = TeaMaster;
    };
    //注ぎかた選択オブジェクト設定
    VOID setPouringSelector( PouringSelector * pouringSelectorObj )
    {
        this->pPouringSelector = pouringSelectorObj;
    };
    //注ぎボタンオブジェクト設定
    VOID setPourButton( PourButton * pourButtonObj )
    {
        this->pPourButton = pourButtonObj;
    };
    //お湯注ぎタスク
    VOID pouringTask( INT StaCd );

};
#endif //__PURING_CONTROLLER_H
```

れたときには，このG-2000システムドメインの修正だけで変更ができます．

これは，デザインパターンのファクトリパターンという考え方に似ています．G-2000システムは，システム全体が使用する部品を供給する工場で，どの工程がどの部品を使用するかを把握しています．

実際にソースコードで説明すると，オブジェクトを生成している部分は，GomaPot.h（**リスト3**，p.117）の「データメンバ」とコメントの入っている部分です．ここで使用するおもなオブジェクトをprotectedのデータメンバとして生成しています．温度監視のオブジェクトは，monitorTemperatureObjとして生成されています．実際の温度制御のオブジェクトはPIDControlObj，onOffControlObjとして生成されています．これらを関連付けるのはGomaPot.cpp（**リスト4**）のcreate()関数です．「TemperatureMonitor（温度監視）の初期化」とコメントの入っている部分で，monitorTemperatureObjのsetBoilControl()という関数で加熱時の温度制御方式を，setSterilizeControl()で殺菌時の温度制御方式，setKeepControl()で保温時の温度制御方式を設定しています．

G-2000システム自体のクラス，GomaPotクラスのオブジェクトは，getGomaObj()というグローバル関数（GomaPot.cpp：**リスト4**）の中で生成します．getGomaObj()関数内部のstaticなオブジェクトとしてgomaPotObjectを生成することで，初回のgetGomaObj()の実行でオブジェクトが生成され，2回目以降の呼び出しでは初回の呼び出しで生成されたオブジェクトのポインタを返すことができます．利用する側は，getGomaObj()を使用してシステムオブジェクトを取得します．

17 タスクの入り口について

タスクの入り口は，グローバルな関数としてGomaPot.cpp（**リスト4**）で宣言しています．お湯注ぎはpoureMain()，水温の監視はmonitorTempMain()です．OSはμITRON仕様のものを想定しています．タスクの入り口は関数で，OSが管理するタスク定義用のテーブルに登録して使用する方法を考えています．

この規模の組み込みシステムではRAMの使用量も制限されるため，動的にメモリを確保するような機能は使用しないようにしています．また，動的にタスクを生成するcre_tsk()システムコールをサポートしていないOSも多く存在するため，タスクの入り口はグローバルな関数としています．そこからGomaPotクラスのpouringTask()，controlTempTask()を経由してPouringControllerクラス（PouringController.h：**リスト5**）のpouringTask()，TemperatureMonitorクラス（TemperatureMonitor.h：**リスト2**）のmonitorTemperatureTask()関数で実際の処理に到達できます（**図6**）．

なぜ，こんなにいくつもの関数を経由しているのかというと，タスクの入り口のグローバル関数ではG-2000システムドメインであるGomaPotクラスしかさわれないようにしているからです．そうすることによって，内部の処理の変更が発生しても，その変更はG-2000システムドメインの外側にまで及ばないで済

〔リスト6〕TemperatureMonitor.cpp

```
////////////////////////////////////////////////////////////
//  ファイルの説明：温度監視クラス実体
////////////////////////////////////////////////////////////
#include "Rtos.h"
#include "TemperatureMonitor.h"

////////////////////////////////////////////////////////////
//  クラス：温度監視クラス
//  機  能：コンストラクタ
//              変数の初期化
//  引き数：なし
//  戻り値：なし
////////////////////////////////////////////////////////////
TemperatureMonitor::TemperatureMonitor()
{
    this->temperetureCondition = TEMP_COND_BOIL;
                                        //初期状態を加熱状態とする
    this->reboilFlag = FALSE;
}


////////////////////////////////////////////////////////////
//  クラス：温度監視クラス
//  機  能：水温の監視と制御
//              はじめに湯を沸かし殺菌した後保温に移る
//              基本的には電源が入っている間は常に保温をおこなう
//              「沸騰」ボタン押下時，蓋を閉じた時に再度沸かし直す
//  引き数：未使用
//  戻り値：なし
////////////////////////////////////////////////////////////
VOID TemperatureMonitor::monitorTemperatureTask( INT StaCd )
{
    for( ; ; )
    {
        this->temperetureCondition = TEMP_COND_BOIL;
        this->boil( STERILIZE_TEMP );
                                        // はじめは 100℃に加熱する
        this->temperetureCondition = TEMP_COND_STERILIZE;
        this->sterilize( STERILIZE_TIME ,
                                  STERILIZE_TEMP );
                                        // 3 分間 100 ℃にする
        this->temperetureCondition = TEMP_COND_KEEP;
        this->keep(  );                                 // 保温
    }
}


////////////////////////////////////////////////////////////
//  クラス：温度監視クラス
//  機  能：加熱
//              加熱の目標温度に達したら抜ける
//  引き数：目標温度
//  戻り値：なし
////////////////////////////////////////////////////////////
VOID TemperatureMonitor::boil( SHORT Target )
{
    this->boilControl->setTargetTemperature( Target );
                                                // 目標温度設定
    SHORT diff;
    do
    {
        diff = this->boilControl->control();  // 温度制御
    }
    while( diff );                       // 目標温度に達したら抜ける
}

////////////////////////////////////////////////////////////
//  クラス：温度監視クラス
//  機  能：殺菌
//              指定時間の保温を行う
//  引き数：時間
//              維持する温度
//  戻り値：なし
////////////////////////////////////////////////////////////
VOID TemperatureMonitor::sterilize( SHORT Time, SHORT Target )
{
    this->sterilizeControl->setTargetTemperature( Target );
                                                // 目標温度設定
    SHORT Period = this->sterilizeControl->getControlPeriod();
    for( SHORT i = 0; i < (Time / Period) ; ++i )
    {
        this->sterilizeControl->control();    // 温度制御
    }
}


////////////////////////////////////////////////////////////
//  クラス：温度監視クラス
//  機  能：保温
//          設定温度を維持し続けるが節電時刻、再加熱の要求があった時は抜ける
//  引き数：なし
//  戻り値：なし
////////////////////////////////////////////////////////////
VOID TemperatureMonitor::keep(  )
{
    this->reboilFlag = FALSE;
    while( !(this->reboilFlag) )
    {
        if( this->powerSaverObj->isPowerOffTime() == TRUE )
                                        // 電源を切る時間かを調べる
        {
            break;
    // 節電時間になったら保温をやめる
        }
        SHORT Target
            = this->temperatureSelectorObj
                ->getTargetTemperature();    // 目標温度を取得する
        this->keepControl->setTargetTemperature( Target );
                                                // 目標温度設定
        this->keepControl->control();              // 温度制御
    }

    // 節電時間中は再加熱に行かないようにする
    while( this->powerSaverObj->isPowerOffTime() == TRUE )
                                        // 電源を切る時間かを調べる
    {
        this->heaterControllObj->StopHeating();
                                            // ヒータを停止する
        dly_tsk( MONITOR_CONTROL_PERIOD );
    }
    this->reboilFlag = FALSE;
}

////////////////////////////////////////////////////////////
//  クラス：温度監視クラス
//  機  能：沸かし直し
//              沸かし直す時に実行する
//              保温の温度制御を中断するためのフラグを設定する
//  引き数：なし
//  戻り値：なし
////////////////////////////////////////////////////////////
VOID TemperatureMonitor::reboil(  )
{
    this->keepControl->breakWaitControlPeriod();
    this->reboilFlag = TRUE;
}
```

みます．タスクの入り口や初期化など，ドメイン構造図の外側からこのシステムに入るところは，すべてGomaPotクラスを経由するようにしています．具体的に，GomaPotクラスオブジェクトを取得するには，先ほどのgetGomaObj()を使用することにより可能ですが，その内部で生成している湯の注ぎアプリケーションドメインや湯沸かし，保温アプリケーションドメインに所属するオブジェクトはprotectedで宣言されているため，参照することはできません．

18 湯沸かしの実装

まず，湯沸かしタスクのメインとなる部分を考えます．この

〔リスト7〕**PowerSaver.h**

```
/////////////////////////////////////////////////////////////
//  ファイルの説明：節電の基本クラスヘッダ
/////////////////////////////////////////////////////////////
#ifndef __POWER_SAVER_H
#define __POWER_SAVER_H

#include "Typedef.h"
#include "RealTimeClock.h"

/////////////////////////////////////////////////////////////
//  ドメイン    ：節電アプリケーション
//  クラスの説明：節電の基本クラス
/////////////////////////////////////////////////////////////
class PowerSaver
{
////////////////////////////////////////////////////////////
//  定数定義
////////////////////////////////////////////////////////////
protected:
#define       DEFAUALT_ON_TIME    ("05:00:00")
                                       // デフォルトのヒータON 時刻
#define       DEFAUALT_OFF_TIME   ("00:00:00")
                                       // デフォルトのヒータ OFF 時刻

////////////////////////////////////////////////////////////
//  データメンバ
////////////////////////////////////////////////////////////
protected:
    const static ClockTime  defaultOnTime;
                                       // デフォルトのヒータON 時刻
    const static ClockTime  defaultOffTime;
                                       // デフォルトのヒータ OFF 時刻
    ClockTime OnTime;
         // ヒータ OFF 時刻（バックアップ電池付きのRAMに保存される）
    ClockTime OffTime;
         // ヒータ OFF 時刻（バックアップ電池付きのRAMに保存される）


    BOOL powerSaveMode;                // パワーセーブモードか否か

    RealTimeClock * pRealTimeClockObj;   // リアルタイムクロック制御

////////////////////////////////////////////////////////////
//  メソッド
////////////////////////////////////////////////////////////
public:
    //コンストラクタ
    PowerSaver();

    // 時刻クラスを設定する
    VOID setRealTimeClockObj( RealTimeClock * realTimeClockObj )
    {
         this->pRealTimeClockObj = realTimeClockObj;
    };

    // 電源を切る時間かを調べる
    virtual BOOL isPowerOffTime() { return FALSE;};
    // 節電ボタン押下時の動作
    virtual VOID onPowerSaveButton(  ) {};
    // 沸騰ボタン押下時の動作
    virtual VOID onBoilButton(  ) {};

protected:
    // 現在時刻を取得する
    VOID getNowTime( ClockTime * nowTime )
    {
         this->pRealTimeClockObj->getNowTime( nowTime );
    };
    // 現在時刻を取得する
    VOID getNowDate( ClockDate * nowDate )
    {
         this->pRealTimeClockObj->getNowDate( nowDate );
    }
};


#endif //__POWER_SAVER_H
```

タスクの動作は，お湯を沸かしいったん沸騰させた後，殺菌のため3分間沸騰状態を継続し，次に目的の温度を維持します．TemperatureMonitor.cpp（**リスト6**）のmonitorTemperatureTask()が全体を制御していてboil()で沸騰，sterilize()で殺菌，keep()で保温を行います．このboil()，sterilize()，keep()を永遠に繰り返すことで，湯沸かし動作を行います．

boil()はboilControlを使用し，目標温度に達した時点で制御を終了します．boilControlの実体は，GomaPot.h（**リスト3**）のonOffControlObjで，GomaPot.cppのcreate()が関連付けします．onOffControlObjはTemperatureControllerクラスの派生クラスであるOnOffControllerクラス（OnOffController.h）オブジェクトです．ただし，boil()ではこれがonOffControlObjであることは意識せず，基本クラスのTemperatureController（TemperatureController.h：**リスト1**）として操作します．そのため，TemperatureControllerクラスの派生クラスであるPIDController（PIDController.h）クラス（PIDController.h）オブジェクトを設定されても動くし，将来FuzzyControllerクラスが開発されれば置き換えることも可能です．

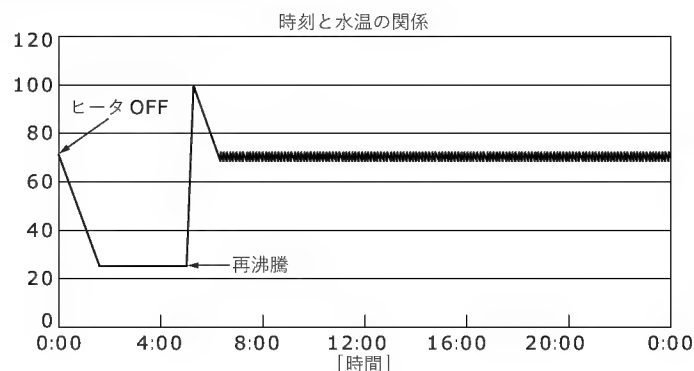
sterilize()はboil()が目標温度に達するまで制御を続けたのに対して，指定時間の制御を続けた時点で終了します．sterilize()が使用しているsterilizeControlも，boil()が使用していたboilControlと同様に，実体はGomaPot.h（**リスト3**）のonOffControlObjです．

keep()は，水温を一定に保つ機能と再加熱要求の有無，節電モードでヒータを切る時間かどうかの判定も行います．メインとなる保温の機能に関してはkeepControlを使用していて，これも実体はGomaPot.h（**リスト3**）にありますが，onOffControlObjではなくPIDControlObjが割り当てられています．

再加熱の判断では，reboilFlagを監視しています．reboilFlagは，reboil()でTRUEになります．ユーザーインターフェースで「沸騰」ボタンの押下を受けたら，reboil()を実行するようにすればいいわけです．reboilFlagがTRUEになると，keep()は処理を中断し終了します．するとmonitorTemperatureTask()がboil()から実行するようになり，再び湯が沸かされます．

節電モードでは，節電モードのヒータOFFの時刻になったらヒータの制御を中断します．この場合はkeep()関数を終了するのではなく，ヒータOFF要求のある間ヒータをOFFにし続けます．ヒータOFFの時刻が終了したらkeep()を終了し，再加熱のときと同様にmonitorTemperatureTask()が再び

〔図7〕水温上昇のようす

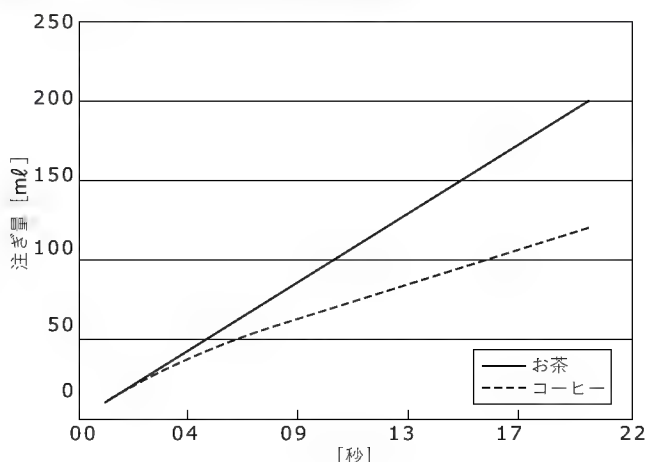
〔図8〕モードの違いによるお湯の注ぎ量

加熱を始めます．

ここで，ドメイン間の結合について考えてみたいと思います．節電モードはドメインが異なっています．たとえば，別のドメインは別のCPUで動かすことができるはずだとか，他のドメインにまったく依存せずコンパイルできるように作るべきだ，といわれてしまうと，弱ってしまいます．keep()の中から他のドメインであるPowerSaverクラス(PowerSaver.h：**リスト7**，p.121)のメソッドを直接呼び出しているからです．

この電子ポットシステムは一つのCPUで動くことが明確になっていますし，システムもそれほど巨大ではないと考えています．そのため，ドメイン間でまったく依存しないように作る必要はないと考えます．むしろ依存をなくすためのしくみを作る工数やソースの追いかけにくさがデメリットになりそうなので，ドメインが別であるということを意識し，依存する部分を明確にしていればよしとしています．また，PowerSaverクラスの仮想関数にデフォルトの処理を入れているので，PowerSaverクラスの派生クラスであるPowerSaverByTimeTable(PowerSaverByTimeTable.h)の開発が終わらなくても，TemperatureMonitorクラスのテストは行うことができるようになっています．ドメインの結合部分がこの程度であれば，ドメインごとにソースを管理することも可能であり，開発工程のずれがあっても対応できます．また，工数も少なくてすむので適度な方法だと考えています．

他のドメインについても同様の考え方で作っているので，とくに解説はしません．ソースリストを参照してください．

19 簡単なテストについて

現状のソースでは，湯沸かし，保温アプリケーションの簡単なテストが行えるようになっています．Waterクラス(Water.h)が，ポットに入っている水をシミュレーションしています．ヒータのON/OFFがgiveHeater()でWaterクラスに設定され，OSのシステムコールdly_tsk()を実行すると，経過時間がgiveTime()により与えられます．Waterクラスではヒータの ON/OFFと経過時間により水温を計算し，現在時刻とともにコンソールに表示します．この際節電アプリケーションも働いていて深夜の時間帯にはヒータを止め，朝方になると自動的に沸かし直します．**図7**がテスト結果のグラフです．午前0：00になるとヒータの電源を切り，朝5：00に再度加熱をします．いったん沸騰した後，設定温度を維持します．

湯の注ぎアプリケーションをテストするには，Rtos.hの「テストのための#define」とコメントの入っている部分の「#define POUR_TEST」の行のコメントを外して有効とし，「#define TEMP_TEST」のほうをコメントアウトします．PouringSelectorクラス(UserInterface.h)の「湯の注ぎ方法取得」とコメントのある部分，getWayOfPouring()の戻り値を変えると，コーヒーとお茶の注ぎ方を変更できます．**図8**がテスト結果のグラフです．

参考文献・URL

1) 組込みソフトウェア管理者・技術者育成研究会(SESSAME)，話題沸騰ポット要求仕様書 http://www.sessame.jp/workinggroup/WorkingGroup2/POT_Specification.htm
2) Enterprise Architect http://www.sparxsystems.jp/ea.htm
3) 渡辺博之，渡辺政彦，堀松和人，渡守武和記著，『組み込みUML〜eUMLによるオブジェクト指向組み込みシステム開発〜』，翔泳社
4) General Principles of Software Validation; Final Guidance for Industry and FDA Staff http://www.fda.gov/cdrh/comp/guidance/938.html

さかい・よしお 組込みソフトウェア管理者・技術者育成研究会(SESSAME)
えとう・よしかず (株)三岩エンジニアリング
いまぜき・たけし (株)豆蔵

第5章 テスト工学からアプローチする組み込みシステムの評価

組み込みシステムのテスト手法

松尾谷 徹

組み込みシステムが高機能化し，複雑化してきた現在，テストの重要性はますます高まってきている．しかし，組み込みシステムのテスト手法については，オープンになっているドキュメントは非常に少ない．そこで本章では，組み込みシステムのテスト手法に関する，さまざまなポイントを解説する．

（編集部）

1 テストの方法について

組み込みシステムのテストについて考えてみましょう．テストを行うには，まずテスト計画についてしっかりした考え方が必要になりますが，組み込みシステムのテスト計画について記述されたテキストはほとんど見当たりません．バンキングシステムやWebシステムと組み込みシステムでは，異なったテストの方法が必要になります．ここでは，組み込みシステムの特徴を，

① **システムの構造**
② **テストの構造**
③ **コンカレント開発**

の三つの視点から整理します．その後，テスト計画策定について説明します．

● システムの構造とテスト

組み込みシステムの種類によって，必要なテストも変わってきます．一言で組み込みシステムといっても，その実体はプラントシステムから，家電やおもちゃまで，あらゆる種類があります．多種多様なシステムが存在するため，組み込みシステムに対する共通的なテストを行うことは困難です．とはいえ，システムの種類ごとにテストを考えるのも実用的ではありません．

そこで，抽象化した組み込みシステムの構造から，どんなテスト活動が必要になるかを考えます．多種多様な組み込みシステムを対象とするので，読者の日常業務では聞き慣れない，少しオーバな表現部分もありますが，第三者の立場になって読んでください．

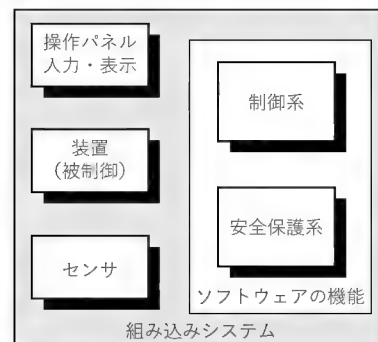
▶安全保護系と制御系

組み込みシステムにとっていちばん困るのは，システムの誤動作や故障時に，利用者や操作者に対して障害が生じたり，装置の破壊，あるいはデータ破壊や不正アクセスが発生することです．電子ポットの例だと，熱湯による火傷の危険性や，ヒータへの過電流による発煙・発火などがそれです．これら非常に困ったことが起らないようにする特性を専門用語では，**安全保護**(**security**)と呼んでいます．

組み込みシステムを概観すると，**図1**に示すように，ソフトウェアも**安全保護系**と**制御系**の二つに分けて構成されます．安全保護系とは，たとえばポットでは，センサが温度の異常を検出したとき，安全を保つためただちにヒータやポンプなどを停止する部分です．制御系とは，通常の機能である，ポットに入ったお湯の温度制御や，ポンプの制御を行う部分です．

ソフトウェアで安全保護系を実装しない組み込みシステムもあります．意外にも原子力分野の多くでは，いちばん信頼性が要求される安全保護系にソフトウェアは単独で使用されていませんし，使用が認められていません．なぜなら，「信用できる」という実績が得られていないからです．

安全保護系に対するテストの要件は，制御系より厳しくなります．「ポットの温度異常でヒータを停止する」という機能のテストは一見簡単ですが，「いかなる条件でもこの機能が動作すること」を考えると難しくなります．「いかなる条件」にはハードウェアの故障や，割り込み禁止時間の長さ，他のタスクとの競合，など無限の組み合わせがあります．一般的に，「いかなる条件」を実機テストすることはできないので，構造的な視点から設計レビューとテストを組み合わせる方法が用いられます．くわしくは後述しますが，テストの方法を考えテスト計画を作成する段階では，安全保護系が含まれるか否かが重要なチェックポイントになります．

〔図1〕組み込みシステムに要求される二つの機能：安全保護系と制御系

▶組み込みシステムの構造について

組み込みシステムがもつ機能や性能をテストによって確認しようとしても，システムはさまざまな特性をもっており，それらを網羅するテストは膨大になり，実機ではテストできない部分もあります．たとえばビルの耐震テストを考えても，実際に完成したビルを使って耐震テストを行うのは不可能です．材料の強度や施工の状況を確認し，構造から推測する以外に方法はありません．

組み込みシステムも同様です．システムテストですべての動作を確認するのではなく，そのうちの一部は，単体テストや結合テストなどによる要素の確認と，システムの構造から推測します．そのため，テスト計画は，組み込みシステムの構造に着目する必要があります．

組み込みシステムの抽象化した構造例を**図2**に示します．構造は，大別してソフトウェアサブシステム，ハードウェアサブシステム，外部サブシステムの三つの要素から構成されると考えられます．これらの三つのサブシステムが有機的に結合し，システムのふるまいが決まります．システムのふるまいとは，機能や性能や安全性のダイナミックな特性のことなのです．

▶外部サブシステム

外部サブシステムとは，組み込みシステムと関係する他のシステムや操作などの総称であり，具体的には外部インターフェース，利用者の操作，温度や湿度などの環境です．外部インターフェースは，組み込みシステムと物理的/論理的に接続され，何らかのプロトコル(protocol：機能単位の動作を決定する意味上および構文上の規則)をもっています．近未来型ポットG-7000の例であれば，給湯情報を通信するためのインターフェースに相当します．利用者の操作とは，組み込みシステムの利用者や運用者が行う操作のことです．通常，利用者は何らかの目的や意図をもって一連の操作を行います．普及型ポットG-2000の例であれば，カップラーメンを作るためにお湯を沸かす一連の操作や，コーヒーをいれる操作などに相当します．テストを行うには，「シナリオ」と呼ばれる利用者が行うと考えられる操作を定義することが必要です．環境とは，温度や電圧など組み込みシステムをとりまく動作環境のことです．ポットの例では，水温，気温，電圧などが環境に属します．当然，これらも定義されないとテストは計画できません．

▶ハードウェアサブシステム

ハードウェアサブシステムとは，組み込みシステムにおけるソフトウェア以外の部分ですが，ソフトウェアとインターフェースをもつ部分に限ってテストの対象として考えます．よって筐体やデザインなどの評価はここでは対象としません．

インターフェースをもつ部分として，操作パネル(入力と表示)，ソフトウェアによって制御される装置，センサなどがあります．また，ソフトウェアが動作するCPUやメモリなども含まれます．操作パネルとは，組み込みシステムに対する操作(入力)や，状態を表示する機能をもつハードウェアです．ソフトウェア側のレシーバやドライバと対になって機能を果します．ポットの例では，水温表示や再沸騰ボタンなどです．

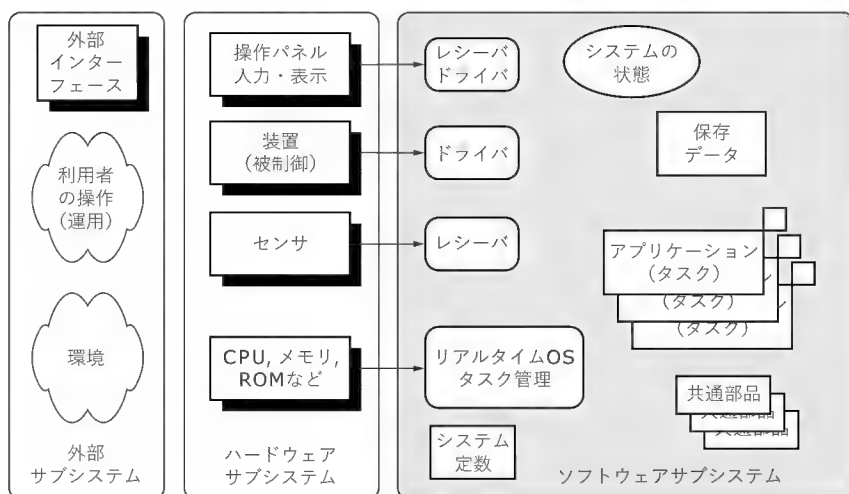
装置(被制御装置)とは，ソフトウェアの制御で動作するハードウェアで，制御するためのドライバソフトウェアと対になって機能します．ポットの例では，ヒータやポンプがこれに相当します．センサは，装置の状態をソフトウェアに知らせるための感知機能をもつハードウェアです．ポットの例では，蓋の開閉，温度，水位，電流などを感知するさまざまなセンサがあります．多くの場合，故障を考慮して複数設置されます．CPUは，組み込みソフトウェアが動作する装置であり，性能をはじめメモリ量，レジスタ数などさまざまな緒元(制限)をもっており，ソフトウェアの性能や応答性に影響を与えます．

▶ソフトウェアサブシステム

ソフトウェアサブシステムとは，組み込みソフトウェアとその動作環境のことです．構造を表す方法はいろいろありますが，**図2**ではリアルタイムOS，ドライバやレシーバ，タスクとして動作するアプリケーション，共通部品などのプログラム部分と，システム定数，保存データ，システム状態などのデータ部分に分けています．

割り込み処理を行い，タスク管理やメモリ管理を行う制御プログラムがリアルタイムOSです．この部分を装置個別に独自開発する例は少なく，市販品や過去に開発したものを使うのが一般的です．リアルタイムOSの動作を決めるパラメータがシステム定数です．

ドライバやレシーバとは，ハードウェアとインターフェースをもち，信号を受信し装置を制御する，対応するハードウェアに固有のプログラムです．ポットの例なら，設定温度を上昇させるスイッチが押されたとき，レ

〔**図2**〕組み込みシステムの構造例

シーバはそれを受け，ソフトウェアが認識する設定温度のパラメータを増加させ，表示ドライバによって，ポットの設定温度の表示を行う一連のプログラムに相当します．

装置の一連の制御やサービス機能を実現するプログラムがアプリケーションです．物理的なタスクやプロセスの構成方法は，使用するリアルタイムOSによって変化します．論理的なモジュールやメソッドは用いる開発方法論によって異なります．ポットの例では，ミルクモードの温度制御や，コーヒーモードのポンプ制御などがこのアプリケーションに相当します．

〔図3〕組み込みシステムにおけるテストの全体像

共通部品とは，用いたリアルタイムOSに備わる共通的なプログラムや，装置に共通するもので，再利用されるプログラムに相当します．それぞれの開発で作成するのでなく，外部から調達するか，既存のものを再利用して使うのが一般的です．

リアルタイムOSのバッファ数や割り込み制御の優先順，あるいはドライバで用いる制御の時間遅れ定数などがデータ領域のシステム定数で，プログラムの動作をカスタマイズするために設定する各種設定値のことです．ポットの例なら，温度を制御するためヒータに通電する時間を制御するパルス幅変調制御のパルス幅を決める定数などが，このシステム定数です．未熟な開発では，このようなシステム定数がプログラムのソースコードの中に埋め込まれてしまうというとんでもない設計もありましたが，この種の定数をシステムレベルで管理するのは設計の常識といわれています．

システムの状態とは，操作パネルやセンサから検出した信号を，ソフトウェアが管理する「状態」を内部表現する変数の集まりです．操作パネルから再沸騰のボタンが押されても，それを受け入れる条件が満たされなければ，プログラムは「保温」から「再沸騰」への状態遷移を行いません．さまざまに設定された条件が満たされているかのチェックを経て，ポットの現在の状態を表す状態変数が設定されます．状態変数は，テストを行ううえで重要な役割を果す変数であり，テスト時に読み出しや変更を加えることにより，テストを合理化することが可能です．保存データとは，そのシステムが保存するデータです．たとえば携帯電話の電話帳や受信記録は消失すると困ります．この種のデータ保護に関するテストは，保存データに関して明確に定義され，構造が分離されていると比較的簡単にできますが，そうでないとテストは困難です．

● テストの構造：全体のテストと部分のテスト

組み込みシステムがもつ機能や性能をすべて一度のテストで確認するのは非現実的です．テストを効率的に進める基本は，独立した部分は小さな部分で確認し，結合テストやシステムテストの負荷を減らすことです．一方，システムの機能や性能を確認するには，分割したテストでは不十分であり，テストもれの問題が生じます．もれには2種類考えられます．

① 確認すべき機能やテスト項目のもれ

組み込みシステムが明示的，暗黙的(安全機能など)にもつ機能をもれなくカバーしなければなりません．

② 考慮すべき組み合わせのもれ

他の機能や環境との組み合わせです．たとえば，安全防護系ではハードウェアの故障との組み合わせが必要になります．そのほかでも，スイッチの同時押下や，割り込みの多発，など，さまざまな組み合わせが生じます．テストの活動(作業)について考えると，単体テストからシステムテストまで，さまざまなテスト活動(作業)があり，さらに，用いる開発環境がターゲットシステムと開発環境で異なる場合など多様です．そこで少し整理し**図3**に示しました．それぞれの活動について，次に説明します．

▶単体テスト

単体テストは，当然ですが開発したメソッドやルーチンに対して行います．再利用した部品や，調達した部品(コンパイラやツールに含まれる関数やライブラリ)に対して受け入れ単体テストを行う例は少ないのですが，自分たちが開発した部品以外のものがあること，しかも，それが大部分であることを認識する必要があります．単体テスト自身は，本番環境(ターゲットマシン上)ではなく，PCなどの開発環境上で行われます．そのとき，使われているマクロや組み込まれる実行時ルーチンの版管理にも注意が必要です．個人で用いる開発環境とチームで用いる開発環境の版が違っていた場合にトラブルにならないようにするためです．

▶結合テスト(ソフトウェア結合テスト)

結合テストは，メソッドやルーチンを結合し，ソフトウェアの機能単位で確認するのが一般的です．このときの入力は**図3**に示したように，単体テストが完了した部品だけでなく，再利用部品や調達部品が含まれることに注意しなければなりません．

テストの実行環境は，多くの場合単体テストと同じ開発環境が使われます．そのため，特定の機能，たとえばリアルタイムOSとの結合テストなどは，シミュレータを準備する必要があります．シミュレータを使うか，本番環境で行うかは，個別の問題としてテスト計画で判断することになります．

▶システム結合テスト

システム結合テストは，システムの要素を結合して行うテストです．組み込みシステム全体から見たソフトウェアサブシステムを結合しテストします．ソフトウェアから見ると，ソフトウェアのシステムテストに相当します．しかし，接続されるハードウェアなどがすべて実機とは限りません．並行してハードウェアのシステム結合テストが行われることがあります．システム結合テストの環境は，一般的にターゲットマシン上で行います．ターゲット環境ではデバッグや開発支援の機能が弱いので，基本的なデバッグを済ませたソフトウェアを結合するのが一般的でかつ合理的です．

▶ハードウェアの結合テスト

システム結合テストでは，ソフトウェアサブシステム以外のハードウェアサブシステム(複数の場合もある)もテストが平行して行われます．注意すべき点は，このときハードウェアドライバソフト(ハードから見たソフトのシミュレータ)が必要であり，それが開発納期や品質に大きな影響力をもつということです．昔はその逆で，ハードウェアが遅れてソフトウェアのテストが遅れることが問題でしたが，ソフトウェアの割合が増え，ソフトウェアによる制御が複雑化した現代では，ソフトウェアができないとハードウェアは評価や調整ができなくなっています．

ハードウェアの結合テストは，所望の性能や出力が得られるようにする調整がおもな目的です．ポットの例なら，実装したヒータの電気特性と，温度を検出する温度センサの特性から，パルス幅変調制御のパラメータ設定を調整する作業です．自動車のエンジン制御などは，この部分のノウハウがいちばん重要であり，多くの時間をかけて最適になる設定を探ります．

▶システムテスト

前述したように，ソフトウェアが組み込まれるシステムは多種多様です．あらゆる種類のシステムがあるため，それらのシステムテストについて共通的に表すことは困難です．ここで対象とするテストも，システムテストすべてではなく，組み込みソフトウェアが影響を与える部分に限ります．おもなテストの視点は，安全系の評価，基本機能の評価，信頼性の評価，利便性の評価，性能の評価です．テストを評価と呼ぶのは，すべてを確認することが原理的に不可能なため，一部を推測し評価するためです．

安全系の評価とは，前述した安全保護系が設計された意図を満たしているか，要求の意図を満たし妥当であるかを評価します．テスト項目としては，安全保護系の機能とその信頼性を評価します．ポットであれば「沸騰状態でポンプを動かしたとき，お湯が飛散しないか」などを調べます．

当該ドメインにとって基本的な機能が動作するか否かを詳細に調べるのが基本機能の評価です．基本機能の評価にはドメインのノウハウが要求されます．自動車のエンジン制御であれば，走行パターンがドメインのノウハウに相当します．ポットなら，使い方パターンとして蓄えられたポットの基本機能の評価です．そのドメインで共通的な部分と，機種個別の部分に分けることができます．

利便性の評価とは，さまざまな付帯機能に対し使い勝手を評価します．具体的には利用者の操作シナリオを抽出し，それを実行して評価します．ポットであれば，実際にコーヒーを入れてみて，かかった時間や，コーヒーの温度など利用者の意図が満たされるか否かを評価します．

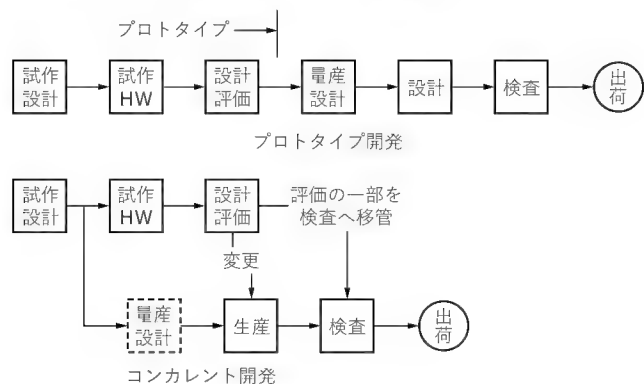
信頼性の評価とは，テスト期間を通して得られたバグ情報と作られたソフトウェアの構造からソフトウェアの信頼性を予測し評価することです．未評価の部分やテスト不足があれば，その部分のテストを追加します．メモリやCPUといったリソース上で，負荷の生成確率からソフトウェアの応答性能や，処理能力を評価するのが性能の評価です．待ち行列の解析など高度な専門知識が要求される評価です．

● コンカレント開発とテストの実装

コンカレント開発は，開発期間を短縮する目的で，相互に影響を与える開発要素を並行して開発する手法です．組み込みシステムで用いられるコンカレント開発は，試作開発と量産設計製造が同時並行に行うケースが多いようです．現在，コンカレント開発を行わないほうが少ないくらい多用されていますが，コンカレント技術の成熟度が低く，問題を多く抱えています．これをテストの視点から整理します．

▶コンカレント開発とは？

ソフトウェア開発に限らず，複雑なシステムの製品開発を1回の設計で誤りなく完全に行うことは不可能です．新製品を開発するにはさまざまな未知の課題があり，試作機を作成し，評価を行ってから量産のための設計を行い，生産に移ります．**図4**のプロトタイプ開発に示すように，評価の工程を経て製品を開発するのが一般的です．試作機を作成し評価してリスクを

〔図4〕プロトタイプ開発とコンカレント開発

低減するのは，設計変更による後戻りコストが大きいからです．ハードウェアが含まれるシステムは，生産に入ってから変更が生ずると膨大な損失が生じ，事業として成立しません．

プロトタイプによる十分な事前評価は失敗コストを低減し，生産性を向上します．しかし，この方式では，新製品を市場に出荷するまでの期間が長くなってしまいます．そこで，開発期間を短縮することに重きを置いた開発方法が必要になります．そのために開発されたのがコンカレント開発です．コンカレント開発は，1960年代の米ソ東西冷戦時代に発展しました．ミサイル開発など国家レベルでの開発競争を勝ち抜くことを目的として，開発期間を短縮するため，試作評価と量産設計を並行して行う方法が用いられました．この開発方法論は，飛躍的に開発期間を短縮できるものの，膨大な失敗コストが生じました．開発に投入するコストに制約がある民間企業がこの開発方法論をそのまま使うのは困難です．開発期間を短縮し，かつ，コストも低減できる方法が求められました．その目的のため，次の二つの技術が開発されました．

① **構成管理と変更管理**：変更と変更によって生ずる影響を合理的に管理し，失敗コストを低減させるための管理技術

② **コンカレント技術**：並行開発において評価の必要な部分を見つけ出し，シミュレーションによる事前評価を行い，変更が容易な構造にするなどの技術

この管理技法と技術を用いて，より複雑なシステムの短納期開発が可能となりました．しかし，コンカレント開発については，製品開発競争を行う企業間において，高度な企業ノウハウがあり，産業界で共有されることはありません．コンカレント開発のノウハウをもたない企業が，市場ニーズから製品開発の納期短縮のため，試作評価と量産設計を並行に行うと，当然，変更や後戻りコストを増加させます．問題は，コストの増加より信頼性の低下にあります．しっかりしたコンカレント開発に必要な技術を導入せず，つまり，コストをかけずに同時並行開発を行うと，結果的には信頼性低下を引き起こし，テスト期間が大幅に増加します．

多くの組み込みシステム開発の現場で，この現象が見られます．計画時のテスト期間と実際のテスト期間が2倍以上かい離する事例がたくさんあります．この場合，テストの技術や能力が低いことだけが原因ではなく，試作開発と量産設計製造の同時並行に対するコンカレント開発のノウハウが不足しており，テスト技術者だけでは対応することが困難です．

▶コンカレント技術について

試作評価と量産など，順序的に意味のある活動や，ハードウェア開発とソフトウェア開発のように相互に影響する活動を平行して実行し，生ずる混乱を緩和させることを目的とするのがコンカレント技術です．相互に影響をもつ活動の課題は，変更による損失をいかに少なくするかにかかっています．変更による損失を少なくするいちばんの方法は，変更を少なくすることです．次が，変更を少ないコストで処置することです．

変更を少なくするには，技術的に未評価の部分を洗い出し，代替手段を用いて評価する方法が使われます．ポットの例なら，実機のハードウェアであるヒータや温度検出器を接続する前に，ソフトウェアでシミュレーションを行い評価することにあたります．またハードウェア側から考えると，新しい温度制御方式で十分な保温機能が動作するかを評価するため，本番のソフトウェアが開発される前に，テスト用のソフトウェアがあると便利です．

変更を少ないコストで処置するには，変更によって生ずるコストについて把握する必要があります．変更によって生ずるコストは，変更の生ずる工程によって異なりますが，一般的にハードウェアのほうが大きく，ソフトウェアのほうが少なくて済みます．よって，ハードウェアの変更に対する備えがソフトウェアに求められます．

ハードウェアの変更に対するソフトウェアの備えは，ドライバ部分とシステム定数で行うことになります．ポットの例では，パルス幅変調制御に対する変更は，時間遅れ係数や感度係数をモジュールから独立し，システム定数とすることによって，独立性を高めることができます．

変更に備えることは，修正のコストより，変更による品質低下を防ぐために行われる回帰テストのほうに注目する必要があります．回帰テストの範囲は影響する範囲に比例します．影響する範囲が不明確なら，すべての範囲で再テストが必要となります．よって，影響範囲を小さくするためモジュールやデータの独立性を高めるのが有効です．

▶構成管理と変更管理

ソフトウェアにおける修正コストの多くは，再テストの範囲によって決まります．よって，変更に対して範囲を特定するための管理技術が必要になります．変更を管理するには，ベースとなる変更前の構成を把握することが必要です．しかし，変更前の構成を把握するのは意外と難しい仕事です．なぜなら，変更が生ずる機能や性能を構成する，ルーチンやモジュールや定数が何であるかを正確に把握しなければならないからです．この特性をトレーサビリティと呼んでいます．

一般的に，構成管理は「ベースライン」と呼ぶ仕切りにより，要求仕様，外部仕様，内部仕様，モジュール（ソースライブラリ），ロードモジュール（製品）を区別します．それぞれの仕切り内で要素は名称やコードによって一意に識別されます．さらに，変更を識別するため，版をもちます．テストの対象として，すべての識別要素を網羅するとともに，版の変更があった場合には，変更された版について網羅する必要があります．

● テスト計画と役割

テストを確実に行うには，必要なテストを網羅することが必要です．必要なテストを網羅するには，どの部分を誰がいつテスト設計を行い，テスト環境を構築し，テストを実施し，改修を行うかについて計画を立てる必要があります．テスト計画は，用いる開発方法論や技法によって変わってきます．たとえば，

オブジェクト指向設計と構造化設計による違い，あるいは，コンカレント開発の程度による違い，などの影響を受けます．どんな状況においてもテスト計画には，テストの対象となるものと，テストのWBS（ワーク・ブレイクダウン・ストラクチャ）と呼ばれる活動の詳細，WBSごとの出力，WBSの作業見積もり，WBSの担当者，スケジュールと納期などを明らかにし，関係者の合意が必要です．

▶必要な活動の網羅

テスト計画では，単体テスト，結合テストなど必要なテスト活動について定義することから始めます．初めての開発でなければ，過去のプロジェクトで行われた活動を評価して用います．**図3**の内容を自分たちの工程と対応付け，作業の枠組みと見通しを立てることが重要です．

▶テスト対象の網羅

今度は，テストの対象となるプロダクトと，そのプロダクトに対応する仕様を明らかにします．単体テストであれば，対象となるメソッドを特定しリストアップします．そして，そのメソッドを定義した仕様書を対応付けます．これを，所望の性能や機能についても行います．

▶スケジュールと役割

仕事とその量が明らかになると，それに必要なリソースの見積もりを行い，実行するチームや担当者をアサインします．同時にできる仕事は限られているし，仕事の順序関係もあります．そこで，時間軸上でスケジューリングしながら計画を立てます．

2 ソフトウェアの内部テストとは？

ここでは，組み込みソフトウェアの内部テストについて説明します．内部テストとは，ソフトウェア内部仕様が正しく実装されていることを確実にするためのテストです．ソフトウェア内部仕様は内部設計によって設計されます．内部設計は，ソフトウェアに対する要求仕様と外部仕様で定められた「ソフトウェアが実現すべき特性」を入力とし，ソフトウェアの構造を設計し，要素を分割します．複数の設計要素（メソッド，関数，ルーチンなど）は，詳細化されソースコードで記述し，マシンで実行可能な部品へと展開されます．

内部設計が構造と要素を作り出すことから，内部テストは要素に対する単体テストと構造に対する結合テストに分けることが一般的です．現実問題として，すべての要素が開発されるのではなく，再利用や外部からの調達が増加しています．再利用部分や調達部分の単体テストを行うことはまれであり，それを補う意味で結合テストが強化される必要があります．

● テストの方法

実際のテストについて説明します．

▶テストの活動

テストを活動（アクティビティ）の側面から考えると，テストを行う直接的な活動と管理活動の二つに大別できます．テストの直接的な活動をさらに分解して考えると，**図5**に示すような四つの活動に分けられます．

① テスト項目の設計

テストの最小単位は，テスト項目と呼ばれます．テストは，必要な数のテスト項目をあらかじめ準備する必要があります．必要な数だけテスト項目を抽出するのが，テスト項目の設計です．ここで必要なテスト項目がもれてしまうと，後のアクティビティではカバーできない重要な活動です．

② テストの環境構築

テストを行うには，単体テストでは，ドライバやスタブなどが必要になります．結合テストでは，シミュレータと呼ばれるテストを行うための環境を準備する必要があります．これらテストに必要な環境を準備する活動です．

③ テストの実行と判定

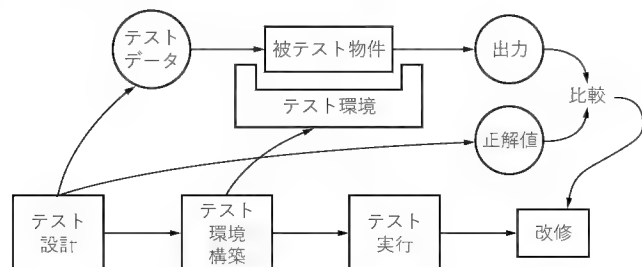
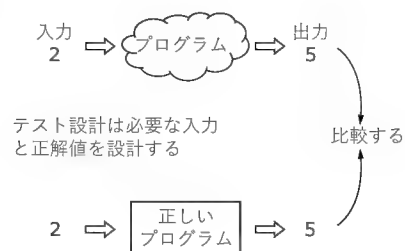
被テスト物件にテスト入力を与え，出力を取り出し，その出力が正しいか否かを調べる活動です．出力が正しいか否かは，テスト項目の設計において，テスト入力とともに予測正解値を設計しておく必要があります．やみくもに入力を与えても，出力結果が正しいか否かはわかりません．

④ 改修

テストの結果出力された値と予測正解値を比較し，不一致の場合に欠陥箇所を探り改修します．欠陥箇所はプログラムに限らず，テスト設計が誤っている場合もあります．

▶テストの設計

テストを行うには，何らかのテスト入力と，その入力に対して確認すべき出力を定義する必要があります．あるプログラムに「2」を入力して「5」が出力されたとします．このときの出力

〔図5〕テストに必要な活動（直接活動）

〔図6〕テストは正解値との比較

「5」が正しいか否かを確認するには，あらかじめ，「2」を入力したとき，どんな値が出力されるかを求める必要があります．よって**図6**に示すように，テスト入力だけでなく，予測結果，あるいは正解値と呼ぶものが必要です．

テストを行うには，テスト入力とその入力に対する正しい出力結果を予測する必要があり，一つの入力と，一つの予測出力のペアをテスト項目と呼んでいます．単体テストであれ結合テストであれ，テストには，必要な数のテスト項目を準備する必要があります．テスト設計では，テスト項目を必要な数だけそろえることが必要です．

思いつきでテストを行うと，出力結果を予測できないためバグがあっても見逃してしまいます．また，必要な数，すなわちすべての組み合わせについてテストすることができません．そこでテスト仕様書に記載し，もれのないようテスト設計を練り上げていく必要があります．テスト仕様書は，確認すべきテスト項目をリストアップし，各テスト項目は，テスト入力とその結果として出力される予測結果を抽出します．

テスト項目を設計するには，「テスト技法」と呼ばれる技法が用いられます．テスト技法は網羅的にテスト項目を設計しますが，二つの尺度で評価されます．一つは網羅性と呼ばれ，もれなくテスト項目を抽出する特性です．もう一つは効率性と呼ばれ，与えられた網羅性の中でいかに少ないテスト項目で済ませるかという特性です．この二つの特性は相反します．良いテスト技法は，少ないテスト項目数で網羅性を高めます．

テストの網羅性も効率性も，テスト設計で用いるテスト技法の影響を強く受けます．後のテスト環境の構築やテストの実行のコストと質を決定するのは，テスト設計で用いるテスト技法といえます．

▶テスト環境の構築

テストを実行するには，被テストプログラムが動作する環境を準備する必要があります．単体テストでメソッドやルーチンをテストしようとすると，ドライバやスタブと呼ばれるテストのためのプログラムを準備する必要があります．

また結合テストでは，用いるデータベースや，用いるハードウェアの動作を疑似するシミュレータのようなものが必要です．このように，テストを実行するために必要な環境を準備するのがテストの環境構築です．組み込みソフトウェアの場合，開発環境とターゲット環境の二つが存在するため，単体テスト，ソフトウェア結合テスト，システム結合テストとの関係で，**表1**に示すようなバリエーションが発生します．開発者がばらばらに考えたのではむだが生じるので，テスト計画や，さらにテスト計画の指針を与えるガイドなどで定めるのが有効です．

〔表1〕開発・本番環境とテスト環境

テストの種類	開発環境	ターゲット環境
単体テスト	ドライバ，スタブ	
結合テスト	OSなどの動作環境	未完成部分の疑似
システム結合テスト	OS，ハードウェアなどの疑似	デバッグ機能
システムテスト		デバッグ機能

〔表2〕テストの結果（出力）と予測結果の関係

	正しいプログラムの出力（プログラムは正しい）	誤ったプログラムも出力（プログラムにバグあり）
正しい予測結果（テスト設計は正しい）	両者は一致	両者は不一致
誤った予測結果（テスト設計にバグあり）	両者は不一致	不定

▶テストの実行と判定

実際に，被テストプログラムにテスト入力を与え，得られた出力と予測結果を比較し，バグがないかを確認します．確認はテスト仕様書のテスト項目ごとに行い記録を残します．テストを実行した結果，次の二つの場合が考えられます．

① **テストの結果と予測結果が一致した**：プログラムが正しいか，あるいは，テストの設計とプログラムの設計が同じ誤りをもっている

② **テストの結果と予測結果が一致しない**：プログラムかテスト設計のどちらかに誤りがある

①の場合，プログラムとテスト設計が同じ誤りであることを確認できないので，この誤りはテストでは検出でききませんが，その確率は低いものです．防ぐには，設計者とテスト設計者を分けると効果的です．②の場合は，詳細に結果からプログラムの動作を調べ，プログラム側の誤りか，テスト設計の誤りかを判断します．一般的に，テスト側の誤りのほうが少ない傾向があります．理由は，プログラムの設計量と比べ，テストの設計量が小さいためです．

▶改修

プログラム側に誤りがあると判明したとき，その誤り箇所を究明し改修を行う活動です．改修はただプログラムの誤ったところを直すだけではなく，次のことに注意する必要があります．

① 副作用

改修の原因となったテスト項目を再確認するのは当然ですが，問題は，その時点までに消化したテスト項目をキャンセルして再確認を行うかどうか，判断する必要があることです．改修による思わぬ悪影響や副作用をテストするにはたいへんな資源が必要になります．それゆえ，改修，とくに後工程における改修は，バグを正しく修正することと，他への影響や副作用が生じないことを十分考慮し，レビューして確実に行う必要があります．具体的な方法としては，修正前と後の差分リストを作成し，第三者によるレビューなどが効果的です．

② 同種の誤り

見つかった誤りの多くは，同じ誤りを他でも作り込んでいる可能性があります．それゆえ，見つかった誤りを改修するだけでなく，同種の誤りについて，他のモジュールのソースコード

に対しレビューを行い除去すると効果的です．

● テストの管理と品質保証

テストの管理活動には，二つの視点があります．テストを行う直接的な活動の計画や，資源の配置，進捗を管理する活動と，品質を確実にする品質保証です．品質保証の活動は，直接，テストを実施，その進捗を管理する活動とは独立して行われます．

▶テストの技法　CFD

テストにとってもっとも重要な活動は，もれなくテスト項目を設計する活動です．もれなくテスト項目を設計するとテスト項目数はどんどん増えます．テスト項目が増えると，テストに費やすコストや時間が増加します．そこで，合理的にテスト項目数を少なくすることが求められます．この相反する二つの要求，品質面から網羅性を高めるため項目を増やす要求と，コストを削減するためテスト項目をコンパクトにする要求のバランスをいかにとるかが重要です．これを解決するために開発されたのが，**CFD**（**Case Flow Diagram**）と呼ばれるテスト技法です．ここではCFDを使ってテスト設計を説明します．

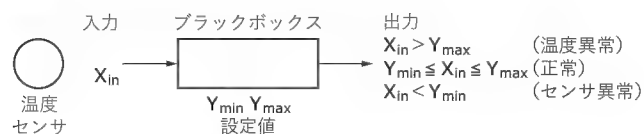
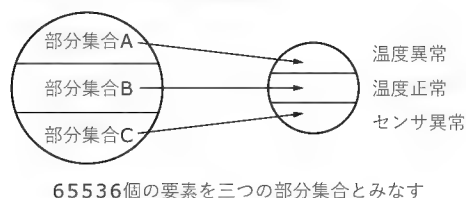
▶テスト項目が爆発する三つの原因

テスト設計において，テスト項目の数が爆発的に増加する原因は，次の三つの場合です．

① 一つの入力が取り得るすべての値
② 複数の入力間の組み合わせ
③ 状態遷移をもつ組み合わせ

CFDは，この三つの原因に対して，テスト項目の優先順位の考え方から，ランク付けを行い，ランクの高いテスト項目を先に実施します．

▶一つの入力が取り得るすべての値

非常に単純なプログラム，たとえば，一つの引き数からなるサブルーチンやメソッドについて考えてみます．**図7**に示すようなブラックボックスとして考えます．ポットの温度センサを監視し，異常温度に対してアラーム信号を出力するレシーバ処理についてテスト項目を考えます．このプログラムは，ポットに設置された温度センサからA-Dコンバータを通して得られた，温度を示す値X_{in}が入力として与えられます．X_{in}：ある設定された範囲，下限を示すY_{min}，上限を示すY_{max}の内にあれば出力は「温度正常」，Y_{max}より大きいと「温度異常」，Y_{min}より小さいと「センサ異常」の信号を出力します．

このサブルーチンの入力値X_{in}が16ビットの整数値であるとすれば，入力の取り得るすべての値は，0から65535までの65536個に達します．このルーチンを網羅的にテストするなら，取り得るすべての値である65536個のテスト項目を作成すれば，ほぼ完全なテスト項目を設計できます．

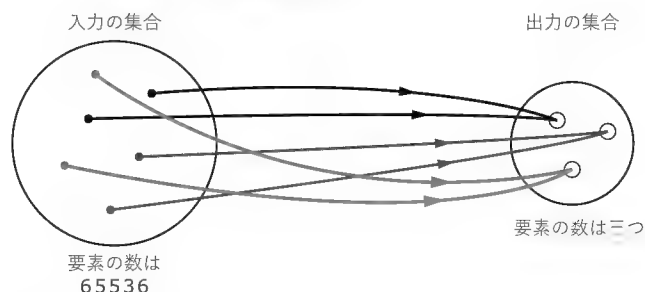
このサブルーチンの出力は，「センサ異常」，「温度正常」，「温度異常」の3種類です．入力との対応は，「センサ異常」が$0 \leqq X_{in} < Y_{min}$，「温度正常」が$Y_{min} \leqq X_{in} \leqq Y_{max}$，「温度異常」が$Y_{max} < X_{in} \leqq 65535$です．この関係を入力の集合と出力の集合の対応と捉えると，**図8**のようになります．

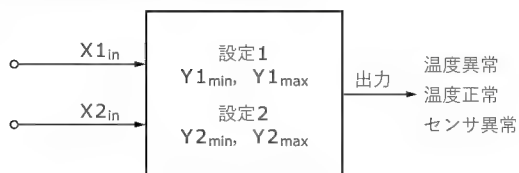
入力集合に存在する65536個の入力要素すべてを等しくテストをするのはたいへんなので，同じ特性をもった部分集合にまとめて考えることにします．**図9**に示すように，三つの出力に対応する入力の部分集合を考えます．この考え方によって入力の集合は，重複しない三つの部分集合に分割することができます．重複しない部分集合を同値類と呼ぶことから，入力集合を重複しない部分集合に分割することを同値分割と呼びます．

CFDでは，同値に対してさらに「入力のすべての要素を排他的に分割する」条件を付与し，テストの網羅性を配慮しています．この例の同値は，入力の集合は三つの部分集合に排他的に分割されています．入力を構成する要素として，マクロに考えると三つの同値があり，さらに詳細に考えると65536個の要素が考えられます．テストも同様に，まず三つの同値について行うことを優先します．

〔図7〕一つの入力をもつブラックボックス

〔図8〕入力集合と出力集合との対応

〔図9〕入力集合の同値分割

〔図10〕同値間の境界値

〔図11〕2入力のブラックボックス

〔表3〕2入力センサ処理の仕様

	X1センサ異常	X1温度正常	X1温度異常
X2センサ異常	センサ異常	センサ異常	温度異常
X2温度正常	センサ異常	温度正常	温度異常
X2温度異常	温度異常	温度異常	温度異常

〔図12〕10入力センサ処理の流れ図

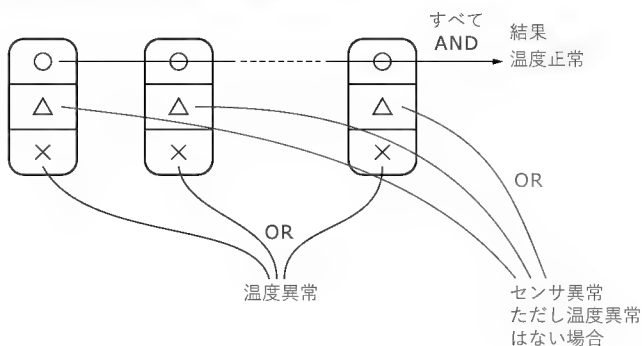
実際に同値のテストを行うことは，同値内から要素を選択することになります．この選択の基準として，境界値分析が用いられます．境界値とは，同値間の境界を意味し，**図10**に示す例の場合では，0，Y_{min-1}，Y_{min}，Y_{max}，Y_{max+1}，65535などに対応します．よって，六つのテスト項目を抽出できます．

▶複数の入力の組み合わせ

今度は，複数の入力からなるブラックボックスについて考えます．**図11**に示す簡単な例で，二つの温度センサからなる温度検出のレシーバ処理を対象とします．この処理は，二つの入力$X1_{in}$，$X2_{in}$と，$Y1_{min}$，$Y1_{max}$，$Y2_{min}$，$Y2_{max}$，から構成されます．

二つの入力に対する仕様は，**表3**に示すような3×3のマトリクスで表されます．それぞれの入力の取り得る値は，65536個であり，正確には4,294,967,296（約42億）です．**表3**はすべての組み合わせではなく，同値間の組み合わせを表しています．この場合のテスト項目は，3×3の九つとするのが一般的です．それでは，マトリクスの要素が増えた場合にどうすればよいのでしょうか．今度は入力の数が10個になったとします．入力同値の組み合わせで網羅しようとすると，3×3×……＝3の10乗で59049個（約6万個）のテストが必要になります．これではたいへんです．どうすればよいでしょうか？

この場合，約6万の組み合わせが存在することは事実です．では，なぜ組み合わせすべてについてテストが必要なのでしょうか？　10個の温度センサに関する動作が，相互に影響する可能性があるからです．10番目の温度センサに着目し，$X10_{in}$の入力を$Y10_{min}$，$Y10_{max}$と比較する処理が，たとえば，1番目の温度センサに関する処理において，$X1_{in} < Y1_{min}$と$X1_{in} > Y1_{min}$の影響を受けるかどうかということです．

もちろん，意図的にそのようなバグを作成することは可能ですが，現在のプログラミング言語やOSの構造はモジュール相互の独立性を高めるように作られており，外部変数を誤って上書きするなど，プログラミング規約を逸脱した結果から生ずるバグしか考えられません．プログラミング規約を逸脱したバグを見つけるには，同値の組み合わせ約6万個でも不十分です．

どのようにしてテスト項目を選べばよいのでしょうか？　同値の組み合わせ，すなわち論理によって異なるため，論理を明らかにする必要があります．**図12**に，論理を分析した流れ図を示します．一つ一つの入力は三つの同値，すなわち○：Y_{min}からY_{max}の範囲内，△：Y_{min}より小さい，×：Y_{max}より大きい，から構成されています．

結果は3種類あり，温度正常，センサ異常，温度異常です．この結果と入力同値の関係を線（流れ）で表します．温度正常は，すべてのセンサが○の場合（論理積），温度異常は，センサの一つでも×の場合（論理和），センサ異常は，センサの一つでも△で，かつ，温度異常ではない場合です．

CFD技法の詳細は割愛しますが，このようにして，入力同値と結果を結びつける流れ図を作成し，仕様を明確にします．その後，**表4**に示すデシジョンテーブルにまとめ，論理的な矛盾がないか確認します．デシジョンテーブルは，上下で二つの部分，すなわち原因の部と結果の部に分かれています．原因の部は，入力項目ごとに分割します．入力項目は取り得る同値，ここでは○，△，×のいずれかの値をとります．あるいは論理関数と考え，真を1，偽を－1で表すこともあります．結果の部ではこの表現方法を使っています．空白は任意を示しますが，他の条件から決まることがあります．

テスト項目は，表を縦に通した関係で与えられます．テスト項目1番は，すべての入力が○のとき，結果として温度正常であることを示しています．テスト項目2番から11番までは，一つ入力が×で，他はどんな値であっても，結果は温度異常になることを示しています．テスト項目12番から21番は，一つの入力が△で，かつ×の入力が一つもない場合に，結果がセンサ異常になることを示しています．

表4のデシジョンテーブルで示した21個のテスト項目が，考えられる6万個のテスト項目の代表です．これですべてのバグを除去可能か否かは，プログラムの作り方に依存しますが，限られたテストリソースの場合，先にこの21項目をテストするのが合理的なテストの考え方です．

〔表4〕10入力センサ処理のデシジョンテーブル

	同値＼テスト項目	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
原因	センサX1	○	×										△									
	センサX2	○		×										△								
	センサX3	○			×										△							
	センサX4	○				×										△						
	センサX5	○					×										△					
	センサX6	○						×										△				
	センサX7	○							×										△			
	センサX8	○								×										△		
	センサX9	○									×										△	
	センサX10	○										×										△
	X1～X10は一つも×でない												1	1	1	1	1	1	1	1	1	1
結果	温度正常	1																				
	温度異常		1	1	1	1	1	1	1	1	1	1										
	センサ異常												1	1	1	1	1	1	1	1	1	1

▶状態遷移

ICや論理回路のテストにおいて，組み合わせ回路と順序回路の問題が知られています．順序回路とは，回路の中に状態を保持する機能，たとえばフリップフロップやメモリをもつ回路のことです．組み合わせ回路は，回路への入力に対して，同じ入力が与えられると，以前と同じ結果を返します．しかし，順序回路は，同じ条件の入力を与えても，同じ結果とは限りません．内部に保持する状態によって結果が異なります．

状態遷移をもつプログラム，組み込みソフトウェアの多くは，状態遷移を大量に含みます．状態遷移をもつプログラムのテストは，特別な工夫を行わないとテストの数が爆発します．その理由は，プログラムの動作に影響を与える状態をプログラムの外部から設定したり読み出したりするのが困難であるためです．ICや論理回路ではテスト容易性という尺度で評価し，これを向上させるため，さまざまなテストのための機能が組み込まれています．

このことはソフトウェアでも同様です．状態遷移に関する設計をフォーマルに行い，状態遷移を表す変数やデータベースをデバッグ時に外部参照できるように設計する必要があります．これらの配慮があれば，同値分割とデシジョンテーブルでテスト項目を設計できます．

● 単体テスト

実際の単体テストについて考えます．単体テストの対象は，二つのとらえ方があります．一つはメソッド，サブルーチン，関数などをコーディングすることとペアで行うテストです．もう一つは，結合テストとの関係から，結合テストのテスト項目が関係する要素の積で増加することを防ぐため，機能や制御の単位に分割して行う単体テストです．前者を物理単体テスト，後者を論理単体テストと呼び区別します．

▶ホワイトボックステスト

被テスト物件の内部構造を明らかにし，その構造をすべて網羅するようにテスト設計を行う考え方です．単体テストではプログラムの制御構造に着目し，パス網羅を行うのが一般的です．本来パス網羅は，テストの網羅性を計測する代用特性として提案された計測尺度です．そのため，パス網羅でテスト設計を行うのは困難です．理由は，パスを実行させる入力条件を抽出することと，パスが実行された結果として何を確認するかについて，何も情報を提供できないためです．

▶ブラックボックステスト

内部構造を見ず，与えられた仕様からテスト項目を設計します．必要な数のテスト項目，すなわちテストデータと予測結果のペアを設計します．先にあげた，同値分割，境界値，デシジョンテーブルなどを用いてテストを設計します．

▶複合型テスト

ブラックボックステストは仕様に書かれていることしかテストできません．仕様に書かれていることだけで十分か否かが問題になります．仕様に書かれていない内部処理とは，たとえば，バッファをOSが取得しようとしたが，バッファ不足で取得できない場合，数秒間待つといった処理です．内部処理に対するテストがもれないようにするため，ブラックボックステストを実施するとき，ホワイトボックステストを併用する方法が，複合型テストです．具体的には，網羅性の計測を行い，未網羅の部分から内部処理を検出します．

● 結合テスト

組み込みソフトウェアの場合，結合テストを開発環境で行うか，ターゲット環境で行うかによって，テストの範囲が変化します．一般的に，開発環境のほうが支援ツールが充実していますが，実行環境としては貧弱です．ターゲット環境では，リアルタイムOSやハードウェアとのインターフェースは充実していますが，デバッガなどの支援ツールは貧弱です．開発環境とターゲット環境でどこまで結合テストを行うかは，それぞれの

製品における個別の課題です．

▶機能間の組み合わせ

結合テストにおけるテスト設計は，先に示した複数の入力の組み合わせによって行います．結合の範囲が広くなると組み合わせの要素数が増加し，組み合わせ数は爆発します．そこで結合テストの階層化を行い，組み合わせ数の爆発を防ぐ必要があります．結合テストの階層化とは，テスト項目設計技法の解説で示した，10入力センサ処理を1入力センサ処理と10入力センサ処理に分離することに相当します．同値分割や境界値に関するテストは，1入力センサ処理（下位の結合テスト）で行い，10入力センサ処理のテストは，1入力センサ処理の結果である○：温度正常，△：センサ異常，×：温度異常を入力とみなして，組み合わせの数を削減させます．

● 再利用・調達部分のテスト

多くの場合，すべてを新規に開発することはまれで，既存部分を再利用し，外部から調達した部品が使われます．このような場合のテストをどうするかが問題になります．結論的には，単体テストなど下位のテストは省略できますが，上位の結合テストは充実させる必要があります．上位の結合テストを充実するには，テストの再利用を行うのが合理的です．テストの再利用は，変更部分に対してテスト項目も変更できなければ意味がありません．テストを再利用し変更を吸収するには，先に示したようテスト設計の情報が引き継がれる必要があります．

3 システム結合テスト

● システム結合テストの目的と方法

システム結合テストの一般的な目的は，複雑なシステムを一度にテストするより，テストが容易にできるサブシステムに分けテストの合理化を行うことです．現実の組み込みシステムでは，もっと重要な役割があります．前述したようにコンカレント開発が一般化しています．コンカレント開発の実際の問題として，ハードウェア仕様の凍結をいつどのような判断から行うかが重要です．ハードウェアは，量産設計を終え製造手配が行われた後で修正が生じると，大きな修正コストを発生させます．

一方ソフトウェアの追加や修正は，製品に組み込むROMに焼き込む時点までそれほど後戻りコストは生じません．ハードウェアとソフトウェアの利害関係ではなく，トータルとして開発期間を短縮し，ハードウェアの後戻りコストを低減させることが重要です．

● ソフトウェアの結合テストとハードウェアの結合テスト

完成したソフトウェアをシステムに組み込み，インターフェースを含めてテストを行います．具体的には，ハードウェアや外部インターフェースを接続し，それぞれの機能が動作することを確認します．コンカレント性によって，実機で行うか，シミュレータを使って行うかを選択することになります．ポットの例は，シミュレータを使ったソフトウェアの結合テストです．

また，ソフトウェアが複雑な制御を行うようになったため，ハードウェア側から見た結合テストにソフトウェアが不可欠となっています．この場合，ソフトウェアの完成を待つと納期が長くなるので，ハードウェアのためのソフトウェアシミュレータが求められます．

4 システム評価

● システム評価の方法論

組み込みシステムの種類は多く，それぞれ異なったシステム評価であり，個別に考える必要があります．**図1**に示した安全防護系と制御系を区別して行います．

● 安全系の評価

安全防護系は，それぞれのドメインノウハウが重要になって，きます．ポットの例でも，さまざまなリスク要因が存在し，抽出されたリスクに対して安全防護が機能するか否かを評価します．

● 信頼性の評価

ソフトウェアに対する信頼性（reliability）は，ハードウェアのように「丈夫で長持ち」という特性ではなく，さまざまな入力条件においても規定された機能が動作することの確かさです．機能の数も，その入力条件も膨大な数になるので，それらをい束ねて評価するかがテスト技法の能力にかかっています．具体的には，同値分割やデシジョンテーブルによるテスト項目数の削減です．

● 基本機能の評価

機能性（functionality）は，機能を洗い出さないと確認できません．機能は外部仕様で定義されますが，機能間の組み合わせや，データのバリエーションについて，テスト技法を用いて設計する必要があります．

● 利便性の評価

使用性（usability）は，利用者から見た使い勝手です．モニタとなる人をアサインして評価したり，試作品を内部ユーザーに提供したりして評価します．

● 性能の評価

効率性（efficiency）は，リソースと負荷によって決まります．決まるといっても待ち行列によるモデル化が必要です．モデル化を行わずに，性能限界を求めるのは非常に困難です．モデル化の場合でも，単一要素（シングルプロファイル）に実測は必須となります．

参考文献・URL

1）デバッグ工学研究所　http://www.debugeng.com/

まつおだに・とおる　デバッグ工学研究所コンサルタント/東京理科大学講師

エピローグ　コミュニティに参加し貢献することで，ともにメリットを得る

組み込みソフトウェアエンジニアとコミュニティ

酒井由夫

本特集記事の投稿プロジェクトメンバは，ソースコードで100万行を超える規模の組み込みシステムに対してオブジェクト指向設計をトライする現役のソフトウェアエンジニア，数々の難しいプロジェクトを品質保証の立場から助けながら大学で教鞭もとる現場主義のテストのプロフェッショナル，組み込み系のさまざまな問題を科学で解決するコンサルタント兼ソフトウェア技術のソムリエ的コーディネータ，リアルタイムOSも自作する組み込み実装のスペシャリスト，そして組み込みソフトウェアシステムの分析者の卵でマーケティングも勉強中のプロジェクトリーダである筆者の5人です．

本特集プロジェクトの命題は，組み込みソフトウェアの再利用性を高め，電子ポット商品群のソフトウェア開発を成功させることでした．この命題に対して，今回のプロジェクトメンバはそのスペシャリティをいかんなく発揮し，プロジェクトを成功に導くことができたと考えています．

実際の組み込み機器の開発では，このようなソフトウェアのスペシャリティをもったプロジェクトのメンバのほかに企画担当者や電気系・機械系の技術者，プロジェクトマネージャなどが必要であり，商品を生産するためには生産技術の技術者もプロジェクトに参加する必要があります(**図1**)．

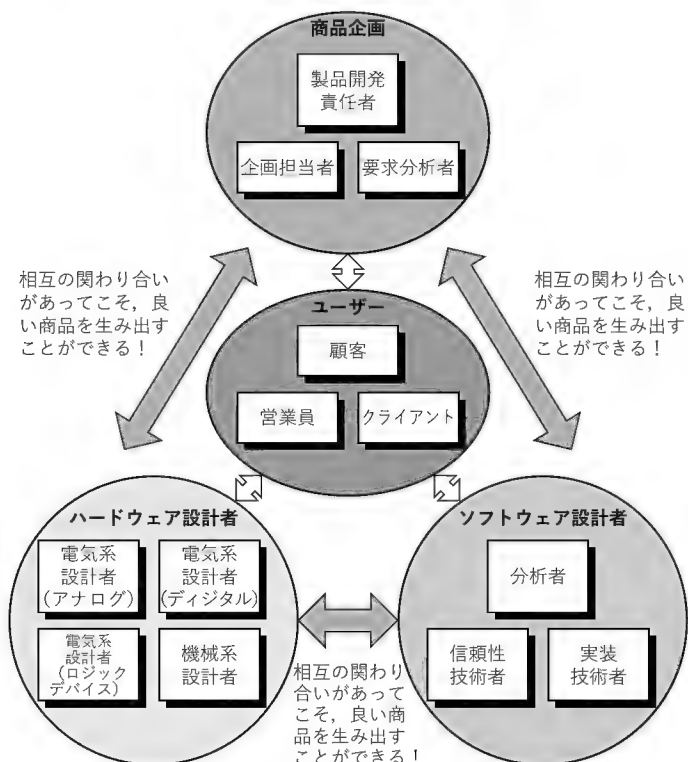
〔図1〕組み込み商品開発プロジェクトチーム

このような実際の組み込み機器開発プロジェクトを成功させるには，プロジェクトの構成メンバの効果的な選抜が成功のカギです．また，プロジェクトリーダは顧客満足度を最大にする商品を作るという目標のもとに「戦略」，「技術」，「チームワーク」がバランスよく効果的に働くように，プロジェクトをマネージメントする必要があります(**図2**と**コラム**参照)．

しかし，プロジェクトメンバの構成についていえば，**図3(a)**のような理想的なプロジェクトが組めることはまれであり，実際の商品開発プロジェクトでは限られたプロジェクトメンバで一人何役かをこなさなければいけないことはよくあります．では，そのようなリソース不足の中でプロジェクトを成功させるためのポイントは，どこにあるのでしょうか？　それは，**プロジェクトのメンバの少なくとも一人が分析者の視点をもつこと**だと筆者は考えます．

組み込みソフトウェア開発の現場では，優れた実装技術者(アーキテクト)が存在していることは珍しいことではありません．なぜなら，組み込みソフトウェアを実際に商品にするためには，ある一定以上の能力をもった実装技術者が必ず必要だからです．しかし，第1章で述べたように，商品に対して要求仕様が多様化しさらに増大してくると，少数の優れた実装技術者だけではプロジェクトを支えきれなくなってきます．現実に組み込み機器開発の現場では，さばききれなくなった要求が実装技術者の両肩にのしかかり，実装技術者がつぶれてしまうという危機さえ起こっていると思います．このような悪循環に入り込んでしまうと，ソフトウェアの品質は劣化し，それを解消しようとして新たに人を投入して品質がさらに下がるという悪循環にいっそう拍車をかけるという状況に陥りかねません．「人月の神話」〔参考文献1)〕は，21世紀に入っても健在です．そのようなときこそ，エンジニアは分析者の視点をもって組み込みシステムを冷静な目で眺めてみることが重要です(**図3**，p.136)．

組み込みシステム商品群の開発では，時間的余裕がないという理由で，過去のソフトウェア資産を「再利用」ではなく「流用」して新しい商品を開発してしまいがちです．そうすると，ソフトウェアの開発スタイルや基本的な構造を抜本的に変えるタイミングがなかなかありません．いい換えると，場当たり的な再利用(＝流用)から体系的な再利用(***プロダクトライン***)への転換を決断するタイミングを見つけることが難しいということです．

Column

顧客満足を第一に考える商品開発（洗濯機メーカーは新しい洗濯機を開発しようとしてはいけない）

10年前，東京近郊に住んでいて地元の電気屋さんではなく秋葉原に家電製品を買いに行く人々がいました．筆者もその一人でした．なぜ秋葉原に洗濯機を買いに行ったのか？　もちろん，秋葉原の電気店では最新の商品が豊富に取り揃えられており価格も安かったということがいちばんの理由でしたが，秋葉原の店員は各メーカーの洗濯機の特徴を熟知していて，自分の要求（予算や使い勝手など）に応じていちばんピッタリ合う洗濯機を選んでくれるだけの知識と販売の経験（洗剤はどれくらい使うのかとか，乾燥機能はついているのかとか，サービス体制は万全かとか）をもっているという理由も大きかったと思います．

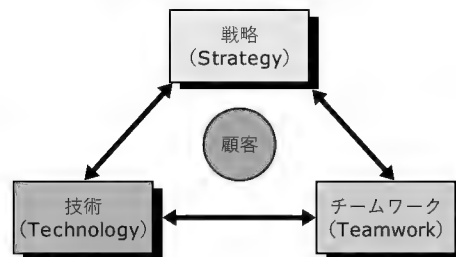
10年後の現在，家電製品はインターネットで激安のものを購入できるようになりましたが，いろいろなメーカーの商品をおしなべて比較したり，誰かにアドバイスしてもらいたいという買い手側の要求はまだあります．その要求にこたえるように，都市近郊では郊外に家電量販店の大規模店舗が出店するようになり，かつて秋葉原の電気店の店員が行っていた販売のノウハウがマニュアル化され，秋葉原へ行かなくても，地元の家電量販店で最新の機種を適切なアドバイス付きで購入できるようになりました．賢いユーザーはこのような家電量販店で商品の知識を仕入れ，インターネットでもっとも安く買える店を探すのかもしれません．

さて，販売側がこのように，商品をただ売るだけでなく，商品＋サービス（商品知識とアドバイス）をセットにして売るようになってきたことを考えると，作り手側はどのようなスタンスで物作りをしていけばよいのでしょうか．

この問題を考えるおもしろい例題として，洗濯機メーカーが新しい商品を開発するときに「どのような洗濯機を作ろうか」と考えてはいけないという話があります．洗濯機メーカーならずともメーカーは新しい商品を企画する際に，現行モデルをベースにして新しい機能を追加しようと考えがちです．なぜなら，現在販売している商品は目に見える「物」であり，その機能や性能について自分たちは熟知しており，顧客からの評判や要望もある程度把握できているので，それをベースに新しい商品を考えることがもっとも自然だからです．

しかし，このようなアプローチには大きな落とし穴があります．なぜなら，洗濯機を使っているユーザーの真の目的は，衣類を洗濯することではないからです．ユーザーの本当の目的は，「汚れた衣服をきれいにすること」です．へ理屈のように聞こえるかもしれませんが「洗濯機で洗濯すること」と「汚れた衣服をきれいにすること」は未来永劫，同等だとはいいきれないのです．

なぜなら，たとえば，自宅近くに，もしも，ワイシャツ1枚を10円でクリーニングするクリーニング屋が現れ，朝玄関脇のボックスに洗濯物を突っ込んでおくと夕方にはきれいになって戻してくれるようなサービスを始めたら，あなたはそれでも洗濯機で洗濯するでしょうか？　もちろん，21世紀になってもそこまでクリーニング料金が安くなるような時代にはなっていませんが，今後絶対にそのようなシチュエーションが現れないとはいいきれません．人々のライフスタイルが変化したり，新しいデバイスが発明されたりすることにより既存の考え方が一変することは，よくあります．したがって，洗濯機メーカーは新しい洗濯機を開発することを考えるのではなく，顧客の要望（この場合は衣服をきれいにするということ）を満足するための商品またはサービスとは何か，また，現在の技術や新しい技術を開発することによってどのような商品またはサービスを提供することができるのかを考える必要があるのです．そうやってユーザー要求を第一に考えた商品やサービスと，現行商品に新しい機能を付け加えたものがイコールであるとは限らないのです．洗濯機自体がいらなくなる時代だってくるかもしれません．

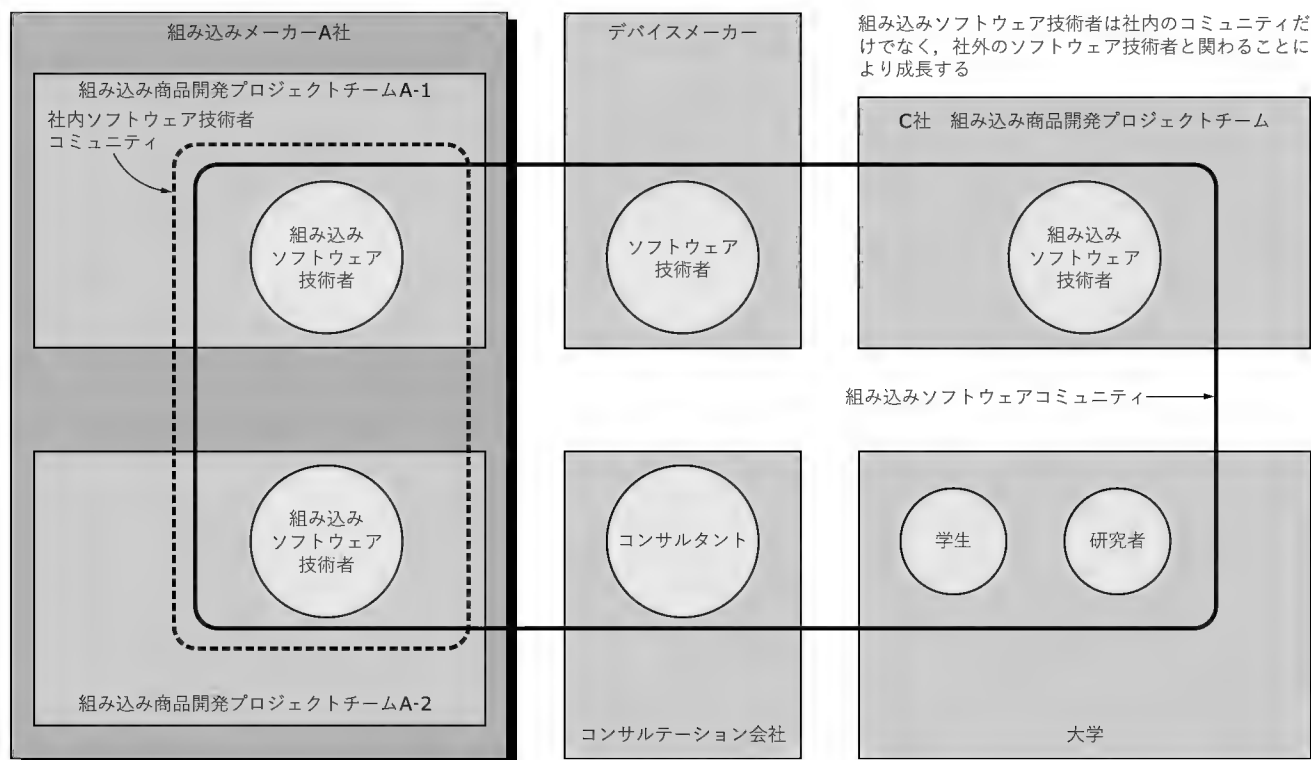
たとえば，フィルムカメラが衰退しディジタルカメラが台頭してくることを10年前に誰が予測できたでしょうか？　この場合は，ユーザーの真の要求は，「記念の映像を残したい」，「きれいな物を画像に残しておきたい」という欲求だったはずですから，カメラを使って撮影したフィルムを現像してもらうという工程から解放され，撮りまくってできのいい画像のみを自分でプリンタを使ってプリントアウトできるようになれば，少々高価でもユーザーはディジタルカメラを買うようになるでしょう．そう考えると，カメラメーカーはカメラを作ることを考えてはいけなくて，「記念の映像を残したい」，「きれいな物を画像に残しておきたい」というユーザー要求を満たすための商品やサービスが，近未来では何になるのかを考える必要があるのです．

本特集では，第2章の***プロダクトライン***と第3章のドメインエンジニアリングで，現在進行中の開発プロジェクトに途中からでも適用できる体系的再利用への転換の方法を示しました．しかし，このような場当たり的な再利用から体系的な再利用への転換（パラダイムシフト）を実行するには，エンジニアが従来の実装技術者の視点で商品開発を眺めていたのでは，実現が難しいでしょう．パラダイムシフトを起こすには，どうしても分析者としての視点が必要になります．荒っぽい言葉でいえば，分析者としての視点と現状のソフトウェア資産を一度バラバラ

〔図2〕「戦略」，「技術」，「チームワーク」

〔図3〕プロジェクト構成の例

(a) 理想的なソフトウェアプロジェクト構成

(b) 最小のソフトウェアプロジェクト構成

〔図4〕組み込みソフトウェア技術者のコミュニティ

にして組み直す，もしくはぶち壊す(！)ような気概とエネルギが必要です．

また，分析者の資質をもったエンジニアはそう多くは存在しないので，実装技術者たちだけでは体系的再利用を実施するためのコア資産の摘出やマネージメントが困難な場合があります．このようなときに有効なのは，分析のスペシャリストによるコンサルテーションや社外コミュニティによる外部からの援助です．外部のコンサルタントを頼むことが難しい場合は，エンジニア同士のコミュニティに参加し，同じ悩みをもつ他の企業のエンジニアたちがどのように問題を解決しているのかを知ることが大事です．同じ悩みをもつエンジニアの存在は，その存在自体が自分たちの追い込まれた心境を幾分やわらげてくれるはずです．

筆者自身，2002年の11月に組込みソフトウェア管理者・技術者育成研究会(SESSAME)に参加し，さまざまな業種の組み込みエンジニアたちと交流することで，自分たちが知らなかったいろいろな解決方法があるのを知ることができました．パラダイムシフトへのきっかけをコミュニティへの参加に求めることは決して悪いことではないと実感しています(**図4**)．

また，要求仕様の実現のために分析のプロフェッショナルに教えを請うことも大事ですが，商品の安全性・信頼性を向上させるためにテストのプロフェッショナルに助けてもらうこともたいせつです．テストは，開発の工程で後回しにされがちですが，テストを科学的，体系的に実施することが，商品の安全性・信頼性に直結することは自明の理だと思います．そのことはどんなエンジニアもわかっていることではありますが，現実には期限のある日程の中でどれだけリーズナブルで効果的なテストを行えるかどうかが問題であり，そこは自分たちだけでは結論が出しにくい領域です．

この問題を解決するためには，現場経験が豊富なテストのスペシャリストの存在はとても頼りになります．限られた時間の中でもっとも有効なテストの手法を提示してもらい，その実績を積み上げて自分たちの業務ドメインにおける体系的なテストの方法をマネージメントすることも，***プロダクトライン***の活動の一つであり，そのようにして構築されたテストの手法は，重要なコア資産となります．

悩めるエンジニアは，自分の悩みを共有できそうなコミュニティに参加し，自分もコミュニティの中での役割を果たしながら悩みを解決する方策を探していくことが，活路を切り開くきっかけになると思います．また，商品開発を行っていく中でさまざまな壁にぶつかったときに，商品開発の向こうにはその商品を待っている顧客が存在することを思い出し，顧客に快適な商品を届けるために乗り越えなければいけない壁であれば，信念をもって壁に立ち向かうことがたいせつであることを忘れないでください．

参考文献・URL

1) フレデリック・P. ブルックス Jr.著，滝沢徹，牧野祐子，宮澤昇訳，『人月の神話－狼人間を撃つ銀の弾はない』，ピアソンエデュケーション
2) 組込みソフトウェア管理者・技術者育成研究会(SESSAME) http://www.sessame.jp/
3) EEBOF－組込み技術者交流サイト－ http://www.bof.jp/eebof/
4) デバッグ工学研究所 http://www.debugeng.com/

さかい・よしお 組込みソフトウェア管理者・技術者育成研究会(SESSAME)

XScaleプロセッサ徹底活用研究

第4回 PCカード/CompactFlashソケットの実装

山武一朗

前回(2003年9月号)は，XScaleに内蔵されているUSBターゲットコントローラを使って，簡単な仕様のUSBターゲット機器を実現した．今回は同じくXScaleに内蔵されているPCカード/CompactFlashインターフェース機能を使って，評価ボードにPCカードやCompactFlashカードを使えるようにソケットを実装する．また，カードを差し込みコンフィグレーションを行うまでの簡易的なカードイネーブラも実装する．(編集部)

はじめに

CQ RISC評価キット/XScaleの評価ボードは，CPUボードを上から見るとボード上部にVMEバス(古いか……)と勘違いしそうなスルーホールが並んでいます．これはPXA250の各ピンが配線されているいわゆるローカルバスで，何らかのバス規格に則ったものではありません．評価ボードに標準で実装されていない機能を拡張するには，このスルーホールから必要な信号線を引き出し，外部にデバイスやボードを接続する必要があります．

PXA250のローカルバスは，組み込みマイコンで一般的なアドレスバス，データバス，チップセレクト，リードイネーブル，ライトイネーブル，ウェイト(レディ)などから構成されます．このローカルバスに一般的なメモリやI/Oデバイスを接続するのは容易です．

しかし，評価目的でさまざまな機能を接続/拡張したいと考えた場合，それぞれローカルバス直結でいくのもよいのですが，何らかの標準バス/インターフェースの切り口を用意することで，拡張性が飛躍的に向上します．

そこで今回は，PXA25xに内蔵されている16ビットPCカード/CompactFlash(以下CFと略)インターフェース機能を使い，評価ボードにPCカード/CFソケットを実装して，PCカードやCFカードを使う事例を解説します．

なお，PCカードやCFカードに関する基礎知識については本稿では説明しないので，カードのアドレッシング方法やアクセスタイミングなどの詳細は参考文献1)を参照してください．ここでは，PCカードとCFカードの違いはアドレスバスの本数のみの違いととらえ，説明しています．そのため，CompactFlashの仕様にあるTrueIDEモードへの対応は考慮していません．

〔図1〕PXA25xのPCカード/CFインターフェース機能のメモリマップ

アドレス	空間
	ソケット1コモンメモリ空間
0x3C00 0000	ソケット1アトリビュートメモリ空間
0x3800 0000	予約
0x3400 0000	ソケット1 I/O空間
0x3000 0000	ソケット0コモンメモリ空間
0x2C00 0000	ソケット0アトリビュートメモリ空間
0x2800 0000	予約
0x2400 0000	ソケット0 I/O空間
0x2000 0000	

1 PXA25xのPCカード/CFインターフェース機能概要

● メモリマップ

図1にPXA25xのPCカード/CFインターフェース機能のメモリマップを示します．PCカードにはコモンメモリ空間，アトリビュートメモリ空間，I/O空間の三つがあり，それぞれ最大64Mバイトのアドレッシングサイズがあります．またPXA25xでは最大2スロット対応なので，それぞれのスロットごとに空間がマッピングされています．

PC/AT互換機では64Kバイトという狭いアドレス空間に，他のリソースとぶつからないようにI/Oアドレスを割り当てなければなりませんが，XScaleではスロットごと，各メモリやI/O空間ごとに独立してメモリ空間が割り当てられているので，他のI/Oデバイスや隣のスロットのカードとのリソースコンフリクトに悩むことなく，カードを使うことができるわけです．

● ソケットインターフェース構成

図2に1スロット時の基本構成例を示します．XScaleには，PCカード/CFインターフェース機能が内蔵されているといっても，PCカードやCFカードのもつすべての信号ピンを直結して「はい完成」というものではありません．図を見るとわかるように，カードがソケットに差し込まれているかどうかの検出や，カードが出力する割り込みは，GPIO機能を使ってCPUに入力する必要があります．

このような構成は，SH-3やSH-4のPCMCIA機能と同等のものです．

図2でわかるように，ソケットに接続するアドレスバスやデータバスは，ローカルバスの信号のそれと同じです．よって活線挿抜対応にするには，間にバスバッファが必要になります．

CE1#やCE2#などのカード制御用のチップイネーブルやリード/ライト信号などの制御信号は，カード用にそれぞれ専用のピンが用意されているので，1スロット時はそれを直結できます．2スロット時はスロットごとに活線挿抜を考えなければならないので，バスバッファを経由する必要があります．

なお，メモリライト信号であるnPWEは，可変長のウェイトを要求するデバイスを接続する場合のライト信号としても使うので，この信号をPCカード/CFインターフェース以外でも使う場合は，1スロット時でもアドレスバスと同様にバスバッファを経由する必要があります．

また，カードからのウェイト信号やI/Oアクセス時のレジスタサイズを示すIOIS16#などの入力方向の信号も，バスバッファが必要でしょう．

見なれない信号としてはPSKTSELがありますが，これはソケット選択信号で，ソケット0をアクセスすると“L”レベルになります．よって1スロット時はそのまま負論理のバスバッファイネーブル信号として使えるわけです．

なお，これらソケット制御用の信号ピンも，GPIO機能と兼用ピンとなっているので，正しくピン機能を初期化しないと，ソケット制御動作をしないので注意してください．

〔図2〕ソケットインターフェースの基本構成例(1スロット時)

● 実用的なインターフェース

図2はもっとも基本的で簡単な仕様のインターフェース構成で，バスバッファの制御や電源のON/OFFもすべてソフトウェアにより制御するものです．カードの抜き差しやカードの割り込み信号(IREQ#)は，PXA25xのGPIOに接続し，GPIOのエッジ検出機能などを活用して割り込み処理を実現できます．

この構成の場合，ソケットの電源ON/OFFもソフトウェア制御になります．よって，たとえばカードを動作中(つまりソケット電源投入状態)で，CPUが長期間割り込み禁止状態になっているときにカードが引き抜かれると，カードが抜き取られたことを知らせる割り込み処理が処理されず，CPUがソケットの電源を切らないうちに，カードが抜かれてしまうことが考えられます．

基本的な設計方針としては，ソケットに電源を入れる操作はCPUからの指示のみで行い，カード未挿入状態を検出したら直ちに電源を切るような回路を実装すべきでしょう．もちろんCPUからの指示で電源を切れるようにすることも必要です．

このように，バスバッファやソケットの電源制御のあたりは，もう少しハードウェア的な仕掛けが必要になるでしょう．

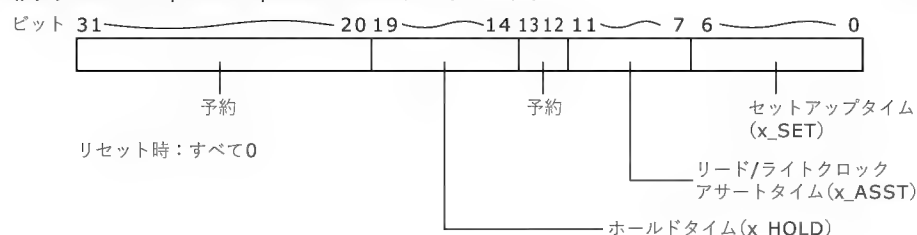
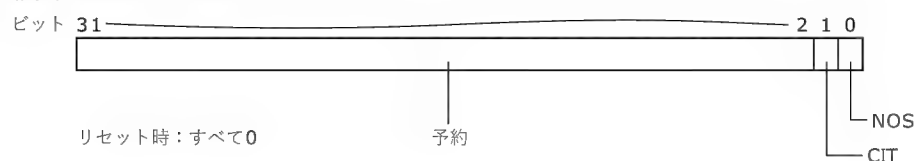
● 制御レジスタ

PCカード/CFインターフェース機能には，アクセスタイミングなどを設定する制御レジスタがいくつか実装されています．

▶ MCMEMx/MCATTx/MCIOxレジスタ

MCMEMxレジスタはコモンメモリ空間の，MCATTxはアトリビュートメモリ空間の，MCIOxレジスタはI/O空間の，セットアップタイムやホールドタイムなどを設定するレジスタです．各レジスタの名前の最後の小文字xは，ソケット0と1の区別になるので，ソケット0のアトリビュートメモリのアクセスタイミングを設定する場合は，MCATT0レジスタとなるわけです．

図3に示すように，ビット19～14がホールドタイム(x_HOLD)，ビット11～7がPCカードからのリード/ライトクロックのアサートタイム(x_ASST)，ビット6～0がセットアップタイム(x_SET)の設定になります．レジスタのフォーマットは三つ(コモンメモリ空間，アトリビュートメモリ空間，I/O空間)とも同じです．よって，たとえばスロット0のアトリビュートメモリ空間のセットアップタイムはMCATT0_SETと表記します．

〔図3〕MCMEMx/MCATTx/MCIOx レジスタフォーマット

〔図4〕MECR レジスタフォーマット

〔表1〕コマンドアサーションコード

MCMEMx_ASST MCATTx_ASST MCIOx_ASST	x_ASST_WAIT	x_ASST_HOLD	x_ASST_WAIT + x_ASST_HOLD
コード値	nPWAIT 認識までのウェイト数（コード＋2）	nPWAIT 認識後のホールド数（2×コード＋1）	コマンドアサートタイム（3×コード＋3）
0	2	1	3
1	3	3	6
2	4	5	9
3	5	7	12
4	6	9	15
5	7	11	18
6	8	13	21
7	9	15	24
8	10	17	27
9	11	19	30
10	12	21	33
11	13	23	36
12	14	25	39
13	15	27	42
14	16	29	45
15	17	31	48
16	18	33	51
17	19	35	54
18	20	37	57
19	21	39	60
20	22	41	63
21	23	43	66
22	24	45	69
23	25	47	72
24	26	49	75
25	27	51	78
26	28	53	81
27	29	55	84
28	30	57	87
29	31	59	90
30	32	61	93
31	33	63	96

セットアップとホールドタイムはメモリクロック(MEMCLK)で何クロック挿入するかをクロック数で指定します．注意が必要なのはリード/ライトクロックのアサートタイムです．これはメモリクロックでのクロック数ではなく，**表1**に示すコマンドアサーションコードを書き込みます．これは基本的にはリード/ライトクロックのアサートクロック数に相当するのですが，リード/ライト信号のアサートを開始してからウェイト信号(WAIT#)を認識するまでのウェイトと，ウェイト解除を認識した後のリード/ライト信号をディアサートするまでのホールドタイムを**表1**に照らし合わせて，該当するコードをレジスタにセットします．つまり，ウェイト認識までの時間とホールド時間をそれぞれ独立に設定することはできません．

▶ MECR レジスタ

NOS ビット(ビット0)は使用するソケット数で，ソケット0のみの1スロットで使う場合は0に，ソケット0/1の2スロットで使う場合は1にします．

また CIT ビット(ビット1)をセットすることで，実際に PC カード/CF インターフェース機能が動作可能状態になります．実際に PC カードにアクセスする場合は，このビットをセットしておかないと，**図2**で示される各空間にアクセスしても，nPCE1 や nPOE といった PC カード制御用の制御線がアクティブにならず，カードにアクセスすることができません(**図4**)．

▶アクセスタイミング

図5(a)に16ビットレジスタにワード(16ビット)アクセスした場合のバスのアクセスタイミングを，**図5(b)**に8ビットレジスタにワードアクセスした場合のバスのアクセスタイミング(I/O アクセス)を示します．**図3**や**表1**の制御レジスタフォーマットの各パラメータが，バスアクセスのどこに関わってくるかを見てください．

図5(b)は，CPU からのワードアクセスに対して，IOIS16# が"H"レベルだったため，ダイナミックバスサイジングにより上位8ビット分を奇数アドレス(MA[0]=1)で再度アクセスを開始しているようすです．この機能が実装されていることにより，接続したカードのレジスタサイズを気にすることなく，同一I/O空間にマッピングすることができます．

2 PC カード/CF ソケットの実装

● 評価ボードのバスバッファ構成

図2の基本構成をもとに，今回設計したインターフェース回路を**図6**(p.142)に示します．今回は CompactFlash ソケットを実

装したので，アドレスバスはA0～A10までの11ビットだけになります．

アドレスバスの上位バスバッファが3ビットだけ，WAIT#などの入力方向のバスバッファが3ビットだけ使っているので，入力方向の信号をバスバッファのA/B方向を逆にしてアドレスバスの上位バッファを使えば，バッファの個数を1個減らせるでしょう．

ソケットの電源制御には，出力制御機能付きの3.3Vレギュレータ REG103（バーブラウン）を使いました．CPUからの電源制御信号nPOWERは負論理で使うので，途中でトランジスタを使ったインバータ回路を入れて論理を反転しています．ソケットの*Vcc*は3.3Vとなります．

なお，本CPUボードの外部拡張端子には，電源として5Vのみが配線されているので，実際にはもう一つ，5V→3.3Vのレギュレータが必要になります．

● カード挿抜検出部

カード挿抜の検出は，CD1#とCD2#がどちらも"L"レベルになったときに挿入されたと判断します．CD1#とCD2#をそれぞれCPUまで配線してもよいのですが，自由に使えるGPIOの本数には限りがあるので，今回は負論理ANDゲートを経由して1本（nCDx）にまとめてCPUのGPIO5に入力します．

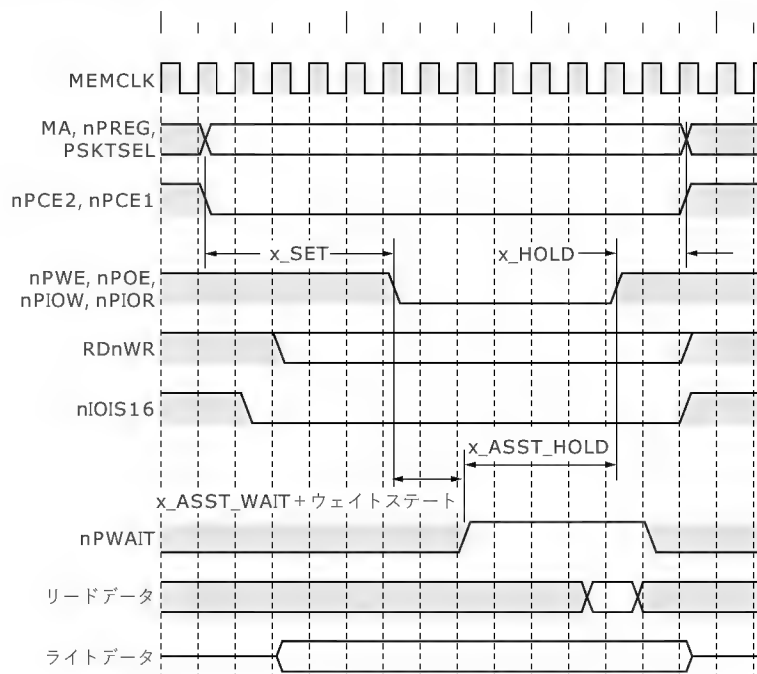
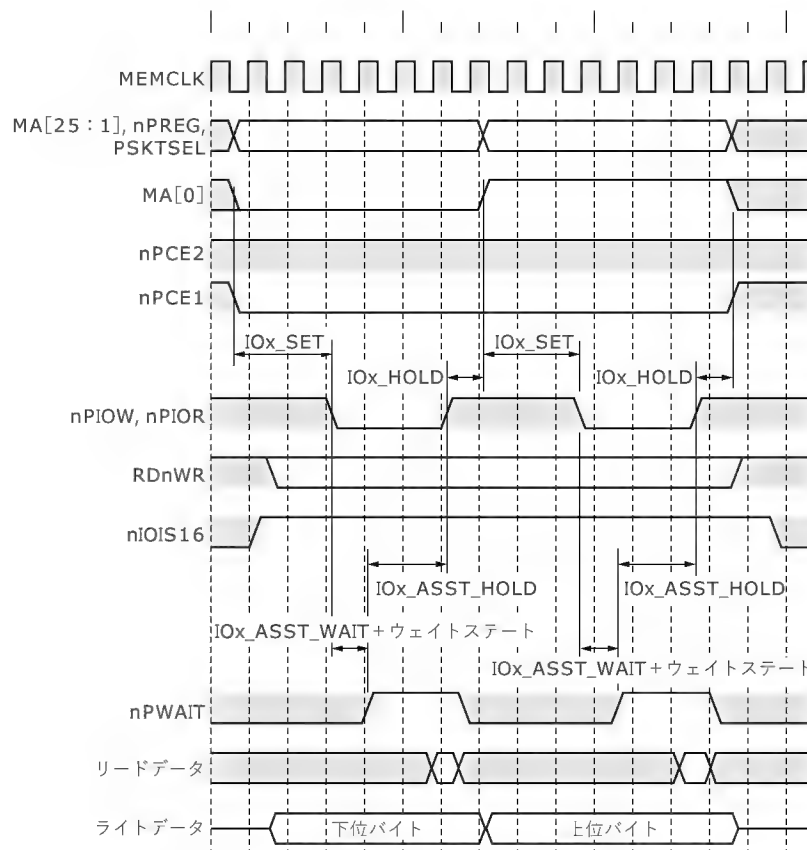
またバスバッファイネーブル（nBUFFER/GPIO6）と電源イネーブル（nPOWER/GPIO7）の各信号も，nCDxと負論理ANDを経由してバスバッファのOEピンや電源レギュレータの制御入力に配線します．

● プルアップ抵抗

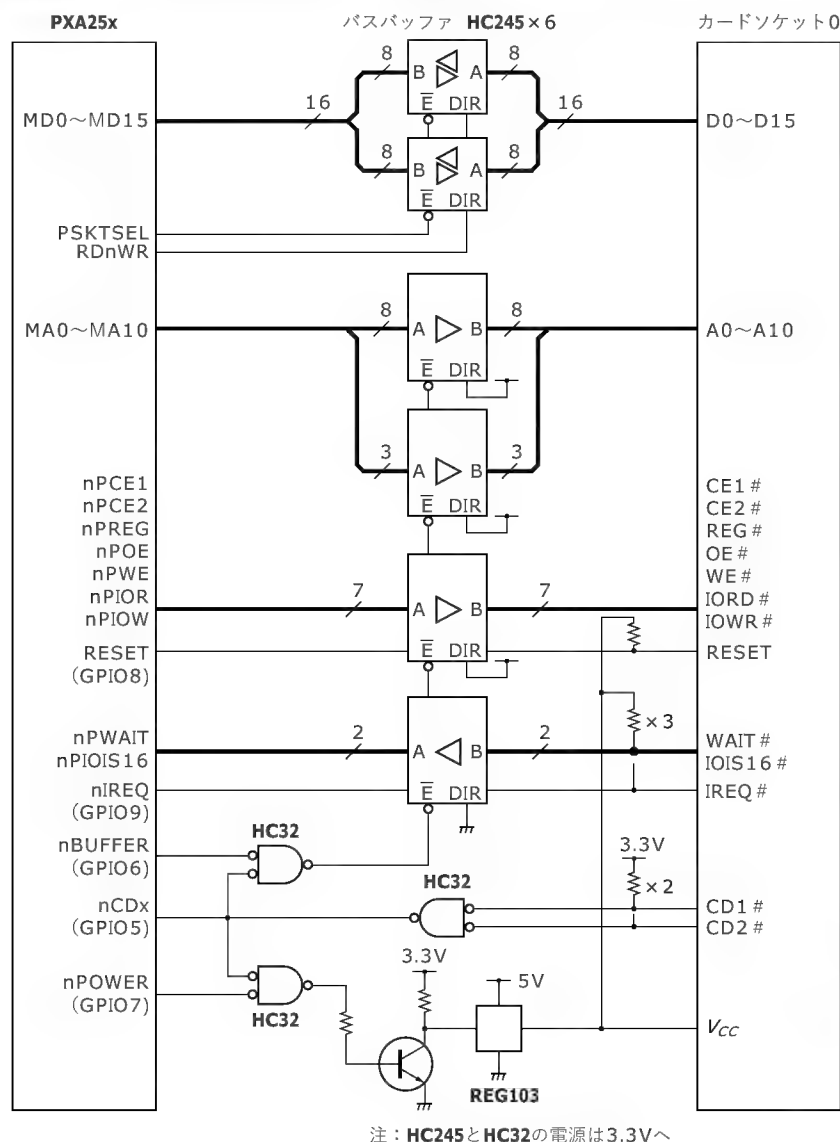
CD1#とCD2#は，カード未挿入時は"H"レベルにする必要があるので，3.3V電源でプルアップします．

しかしWAIT#やIOIS16#，IREQ#の入力信号，そしてRESET出力信号は，ソケット電源である*Vcc*でプルアップしました．ソケット電源を入れずにバスバッファのみをイネーブル状態にすると，入力信号が不安定な状態になるので，バスバッファをイネーブル状態にした後はすみやかにソケットの電源を入れてください．

またRESET信号は，バスバッファがディセーブル状態でソケットの電源のみが入ったときに，カードをリセット状態にするために

〔図5〕バスアクセスタイミング

（a）16ビットレジスタにワードアクセスした場合

（b）8ビットレジスタにワードアクセスした場合

〔図6〕CFカードソケットインターフェースの回路

注：HC245とHC32の電源は3.3Vへ

必要です．

● 未使用信号について

図6の回路図には出てこない信号線について簡単に解説します．なお，これらの未使用の信号のくわしい意味などは，参考文献1)を参照してください．

まずCSEL#は，CFカードをTrueIDEモードで動作させたときに使う信号なので，今回は使用しません．

SPKR#はモデムカードなどでネゴシエーション中の音(「ピーガガガ」というあの音)を，ホストシステムのスピーカから出すときに使います．今回は使用しません．

STSCHG#は，メモリカードとI/Oカードのコンボカードなどで，メモリカード部のバッテリ電圧低下やライトプロテクト状態が変更されたことをホストに示す信号です．今回は使用しません．

INPACK#は，カードがI/Oモードで動作しているときに，カード上でアドレスデコードした結果を出力(ホスト側から見ると入力)する信号です．今回のCFソケットはI/Oカードを使うことを想定していますが，**図1**のメモリマップでわかるように，I/O空間はソケットごとに独立して64Mバイトの空間が確保されているので，ホストシステムの使うI/O空間に割り込ませてI/Oアドレスを割り当てる必要はありません．この信号も使用しません．

もう一つ，VS1#とVS2#はカードの動作電圧を示すものですが，今回は3.3V動作のCFカードと限定しているので，この信号を見る必要はありません．

● バスアクセスのようす

試作したCFソケットボードの外観を**写真1**に示します．さらに**図7**にロジアナで測定したバスアクセスのようすを示します．**図7(a)**がアトリビュートメモリ空間を読み出しているときのアクセスで，**図7(b)**がI/O空間をアクセスしているようすです．実際にはデバイスが存在しない空間なので，IOIS16#がプルアップされている状態となり，8ビットデバイスであると判定され，2回バスサイクルが起こっていることがわかります．

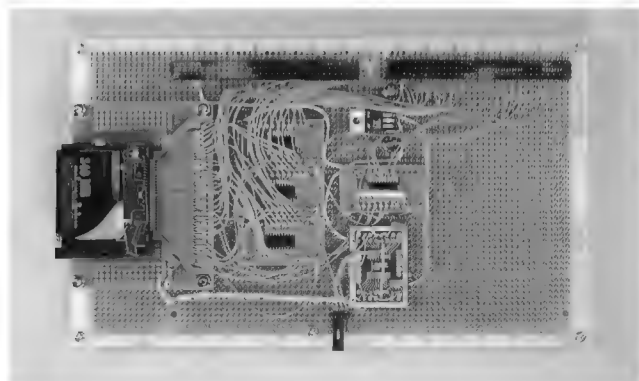
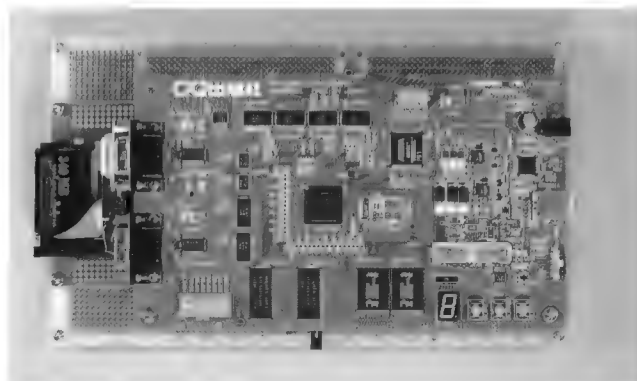
3 カードイネーブラの移植

● カードイネーブラとは？

PCカードやCFカードのI/Oカードは，電源を投入した状態では動作を開始することができません．どのI/Oアドレスで動作させるかを決めて，カードをI/Oモードに設定しなければなりません．この操作を「カードをコンフィグレーションする」と呼びます．

そのためには，そのカードがどれくらいのアドレス空間を必要とするか，割り込みは使うのかといったさまざまなカードの属性情報を読み出して，カードを適切にコンフィグレーションする必要があります．この属性情報を格納しているのがアトリビュートメモリ空間となるわけです．アトリビュートメモリ空間や属性情報のフォーマットなどの詳細は，参考文献1)を参照してください．

簡単にまとめると，アトリビュートメモリ空間にアクセスして属性情報を解析し，カードが要求するリソースを取得して，最適なI/Oアドレスで動作するようにカードをコンフィグレーションするソフトウェアが必要だということです．このソフトウェアのことを，カードを有効状態(イネーブル状態)に初

〔写真1〕試作したCFソケットボード

(a) ユニバーサルボード上にCFソケットボードやバスバッファを実装

(b) 上にXScale評価ボードをスタック接続したようす

期化するソフトウェアということで，イネーブラと呼びます．

ここで作成したカードイネーブラは，基本的には参考文献1)の付属CD-ROMに収録されているカードイネーブラのサンプルプログラム(CARDBUS.C)を改造するかたちで，作成しました．

● PCカード/CFインターフェース制御レジスタの初期化

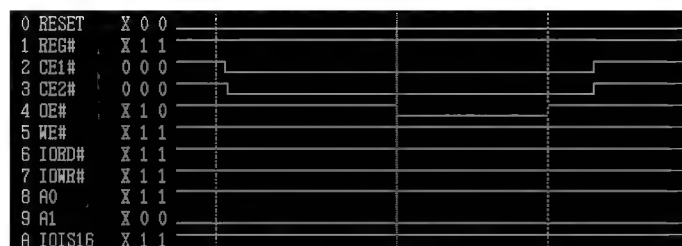
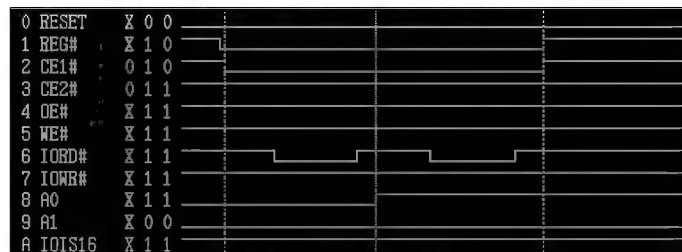
本CPUボードでは外部バスクロックが100MHzとなっているので，1クロック10nsとしてアドレスセットアップやホールドタイムを計算します．

PCカードの仕様では，メモリ空間へのアクセスには速度バージョンによりいくつかの規定があります．アトリビュートメモリの読み出しでは，速度バージョンとして600ns(電源電圧3.3V)と300ns(電源電圧5V)の規定があります．またアトリビュートメモリへの書き込みやコモンメモリの読み書きには，250ns/200ns/150ns/100nsの規定があります．

アトリビュートメモリ空間は，カードイネーブル時にカードの属性情報を読み出したり，カードコンフィグレーションを行う場合に使うメモリ空間です．この空間は初期化時にのみ使う空間なので，アクセス速度は遅くても全体のパフォーマンスには影響を及ぼさないでしょう．

そこで，アクセス時間がもっとも長い電源電圧3.3Vを想定したタイミング(速度バージョン600nsの規定)で，アトリビュートメモリおよびコモンメモリ空間へのアクセスタイミング制御レジスタを初期化しました．アトリビュートメモリ空間とコモンメモリ空間のアクセスタイミングの設定レジスタは独立しているので，コモンメモリ空間のアクセス速度はもっと高速な設定も可能です．しかし，今回想定しているのはI/Oカードなので，基本的にコモンメモリ空間を使うことを想定していません．今回はコモンメモリ空間の設定はアトリビュートメモリ空間の設定と同一の設定としました．

I/O空間へのアクセスは速度バージョンや電源電圧の違いによる規定はありません．PCカードの仕様より，IORD#やIOWR#のアサート幅が最小165ns，アドレスセットアップが70ns，ホールドタイムを30nsとして設定しました．

〔図7〕ロジアナで測定したバスアクセスのようす

(a) アトリビュートメモリ空間の読み出し

(b) I/O空間の読み出し

● GPIOの初期化

すでに説明したように，PCカード/CFインターフェースの制御線はGPIO機能と兼用になっているので，カード制御用に使う信号はPCカード/CFインターフェースとして設定し，ソケットに対しての入出力方向も正しく初期化する必要があります．

さらにカード挿抜検出やカードの割り込みなどをGPIOに接続しているので，これらも初期化しなければカードの検出や割り込みが動作しません．

今回はGPIOのうち，GPIO5をカードディテクト信号として入力方向で，GPIO6をバスバッファ制御信号として出力方向で，GPIO7を電源制御信号として出力方向で，GPIO8をRESET制御信号として出力方向で，GPIO9をカード割り込み信号(IREQ#)として入力方向で使います．

● GPIOの割り込み設定

また，GPIO5のカードディテクト信号とGPIO9のカード割り込み信号は，エッジを検出したら割り込みを出力するように

〔リスト1〕PCカードソケット初期化ルーチン(一部)

```
/* PCカードソケット初期化 */
void PCCard_Init(void)
{
    /* PCカード検出/バスバッファ制御/電源制御/RESET制御/カード割り込み入力 */
    *GAFR0_l = (*GAFR0_l & 0xFFF003FF);    /* GP9~5 GPIO機能 */
    *GPDR0   = (*GPDR0 & 0xFFFFFC1F) | 0x1C0;   /* GP8/GP7/GP6 出力 GP9/GP5 入力 */
    *GPSR0   = 0x1C0;   /* GP8/GP7/GP6 'H'出力 */
    /* PCカードコントローラピン設定 */
    *GAFR1_u = (*GAFR1_u & 0xFFF00000) | 0x5AAAA;    /* GP57~48 PCカードコントローラ */
    *GPDR1   = (*GPDR1 & 0xFC00FFFF) | 0xFF0000;     /* GP48/49/50/51/52/53/54/55 出力 GP57/GP56 入力 */
    /* PCカードコントローラレジスタ設定 */
    *MECR    = 0;
    /* PCカードコントローラタイミング設定 */
    *MCMEMX = 0x0001C4A3;    /* Hold7 / Command9 / Setup35 */
    *MCATTX = 0x0001C4A3;    /* Hold7 / Command9 / Setup35 */
    *MCIOX  = 0x0000C207;    /* Hold3 / Command4 / Setup7  */
}

/* PCカードソケットバッファイネーブル制御
    != 0  イネーブル
    0     ディセーブル
*/
void PCCard_BufferCtrl(int Ctrl)
{
    if (Ctrl) {
        *GPCR0 = 0x40; /* GP6 'L'出力 */
    } else {
        *GPSR0 = 0x40; /* GP6 'H'出力 */
    }
}

/* PCカードソケット電源制御
    != 0  電源ON
    0     電源OFF
*/
void PCCard_PowerCtrl(int Ctrl)
{
    if (Ctrl) {
        *GPCR0 = 0x80; /* GP7 'L'出力 */
    } else {
        *GPSR0 = 0x80; /* GP7 'H'出力 */
    }
}

/* PCカードソケット RESET制御
    != 0  RESET解除
    0     RESET
*/
void PCCard_ResetCtrl(int Ctrl)
{
    if (Ctrl) {
        *GPCR0 = 0x100;     /* GP8 'L'出力 */
        *MECR  = 2;
        *GEDR0 = 0x00000220;
                            /* GP9/GP5エッジ検出フラグクリア */
    } else {
        *GPSR0 = 0x100;     /* GP8 'H'出力 */
    }
}

/* PCカードが差し込まれているかどうか */
int PCCard_DetectCheck(void)
{
    if (*GPLR0 & 0x00000020) {   /* CDx(GP5)の状態' */
        return 0; /* カードが差し込まれていない */
    } else {
        return 1; /* カードが差し込まれている */
    }
}

/* PCカード割り込み初期化 */
void PCCardInt_Init(void)
{
    /* GPIO割り込み初期化 */
    *GRER0 = (*GRER0&3) | 0x20;  /* GP5立ち上がりエッジ検出 */
    *GFER0 = (*GFER0&3) | 0x20;  /* GP5立ち下がりエッジ検出 */
    *GEDR0 = 0x00000220;     /* GP9/GP5エッジ検出フラグクリア */
    /* 割り込みコントローラ初期化 */
    *ICLR &= 0xFFFFFBFF;     /* GPIOx=IRQ */
    *ICCR = 0x00000000;
    *ICMR |= 0x00000400;     /* GPIOx割り込み使用 */
}
```

〔リスト2〕割り込み処理ルーチン(一部)

```
void    SS_Irq(void)
{
    /* GPIO(GP5/9)割り込み発生 */
    if((*ICIP & 0x00000400) == 0x00000400) {

        /* PCカード検出割り込み */
        if (*GEDR0 & 0x00000020) {   /* GP5 エッジ検出 */
            if (*GPLR0 & 0x00000020) {   /* CDxの状態' */
                /* カードが差し込まれていない */
                *MECR  = 0;    /* ソケットインターフェースディセーブル */
                *GFER0 = (*GFER0&3) | 0x20;
                                       /* GP5立ち下がりエッジ検出 */
                PCCard_ResetCtrl(0);     /* カードRESET */
                PCCard_PowerCtrl(0);     /* 電源OFF */
                PCCard_BufferCtrl(0);    /* バッファディセーブル */
                *LEDPORT = led_7seg[0];
                /* カードが抜かれたことを検出した場合は、速やかに */
                /* リセット/ディセーブル状態に設定する */
                CFCard_WakeUpFase=0;
                PCCARD_Enable=0;
                CF_Event=-1;
            } else {
                /* カードが差し込まれている */
                CF_Event=1;
            }
            *GEDR0 = 0x00000020;     /* 検出フラグクリア */
            CF_IntCount++;
        }

        /* PCカードリソース割り込み */
        if (*GEDR0 & 0x00000200) {   /* GP9 立ち下がりエッジ検出 */
            *GEDR0 = 0x00000200;     /* 検出フラグクリア */
            PCCARD_Interrupt();      /* PCカード割り込み処理 */
            CF_Event = 0;
            CF_IntCount++;
        }

    }

    /* RTC割り込み発生 */
    if((*ICIP & 0x80000000) == 0x80000000) {
        Timer_Count++;
        *RCNR = 0x0000;
        *RTSR = 0x0005;     /* 割り込みクリア */
    }
      /* 他に割り込み処理があれば記述 */
}
```

設定しておきます.

カードディテクト割り込みは，カードが差し込まれると“H”から“L”レベルへ，抜き取られると“L”から“H”レベルに変化します．よって，立ち上がりと立ち下がり両方のエッジで割り込みを出力するように設定します.

カード割り込み信号は，通常状態はプルアップ抵抗で“H”レベル，割り込み出力時に“L”レベルという動作を想定して，立ち下がりエッジを検出したら割り込みを出力するように設定します.

以上をまとめて，**リスト1**にPCカードソケット初期化ルー

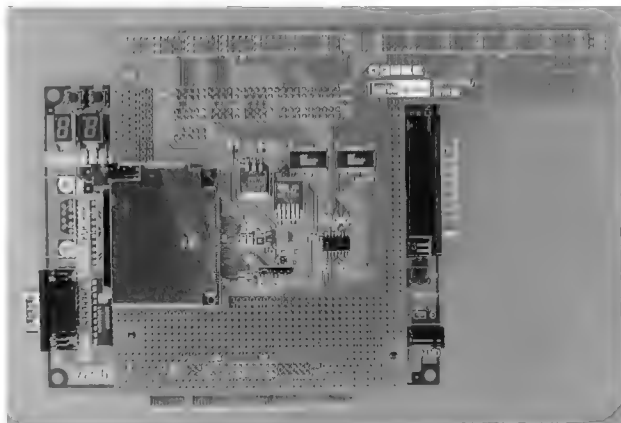
Column 1

CompactFlashソケット拡張キット

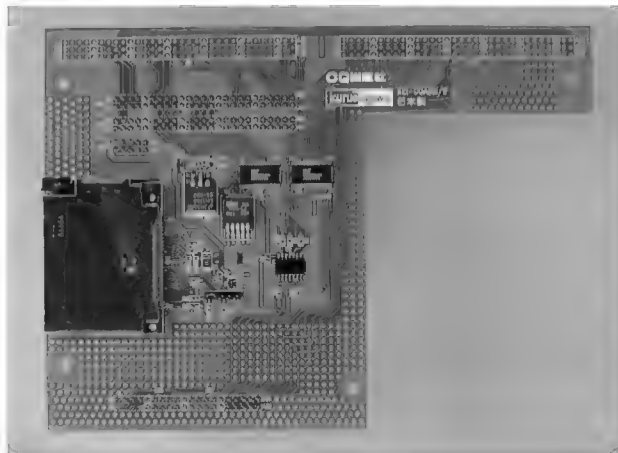
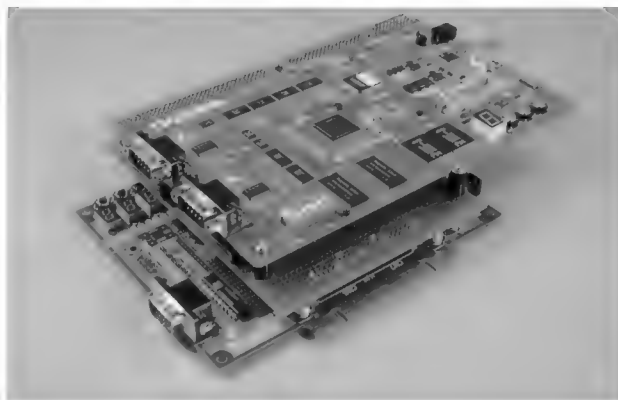
CQ RISC評価キット/XScale(CQ出版)に，CompactFlashソケット(以下CFソケット)やFPGAを接続するための拡張キットが用意されました．

XScale評価ボードにCFソケットを実装した場合は，**写真A**のCFソケット拡張キットの上にXScale評価ボードをスタック接続し，**写真B**のようにして使います．

XScale評価ボードにFPGAを接続する場合，FPGA評価キットとして次の2種類のキットが対応しています．一つはStratix EP1S10(アルテラ)を搭載した「Stratix評価キット」(http://www.cqpub.co.jp/eda/Stratix/)と，Spartan-IIE XC2S300E(ザイリンクス)を搭載した「Spartan-IIE300評価キット」(http://www.cqpub.co.jp/eda/Spartan2e300/)です．どちらでも使い慣れたほうのFPGAを使用することができます．これらFPGA評価キットのどちらかの評価ボードを1枚用意し，その上にCFソケット拡張キットを載せ，さらにその上にXScale評価ボードをスタック接続します(**写真C**)．

〔写真B〕Stratix評価ボードと接続したようす

CF拡張キットは，下記URLに詳細情報を掲載しています．購入方法についても，下記URLを参照してください．

■問い合わせ先
- 来栖川電工(有)
  http://www.kurusugawa-ele.co.jp/cq/cfkit.html
- 組み込みshop　http://shop.kumikomi.net/webmall/

〔写真A〕CFソケット拡張キット

〔写真C〕Stratix評価ボードとXScale評価ボードを接続したようす

チンを，**リスト2**に割り込み処理ルーチンを示します．

PXA25xで特有な処理としては，MECRレジスタのCITビットの設定でしょう．このビットはリセット解除のときにセットし，ソケット初期化時と割り込み処理でカードが抜き取られたことを検出したときにクリアするようにしています．

4 動作確認アプリケーション

- PIO割り込み動作確認プログラム

カードイネーブラだけでは，本当にソケットが動作しているかどうか外から見てもわからないので，実際に何らかの動作をするプログラムに仕上げてみましょう．

筆者は，参考文献1)でPIO入出力に対応した16ビットPCカードを試作しています．これはPCカードですが，**写真2**のようなPCカードをCFソケットで使うためのアダプタを使って，ここで作成したCFソケットに差し込み，PCカードを認識するかどうか，そしてPCカードの割り込みをカウントするかどうかのテストプログラムを作成しました．

参考文献1)のテスト用PIOカードの動作確認部分も，今回のサンプルプログラムでベースとしたCARDBUS.Cに含まれているので，そのまま流用させていただきます．ただし，DOS環境とアーキテクチャが異なる部分があるので注意が必要です．

〔リスト3〕PIO PCカードの割り込みカウントプログラム

```
/* PCカード割り込み処理 */
void PCCARD_Interrupt()
{
    unsigned char *p;
    p=(unsigned char *)PC0IO;
    *(p+PCCard_IOaadr+4)=1; /* 割り込みクリア */
    CFCard_Int++;
}


/* 挿入されたCFカードをチェック */
void Init_CFCard(void)
{
    unsigned int CCR_Address;
    unsigned char CfgBuff[256],CardName[256],*p;
    CardName[0]=0;
    CardName[1]=0;
    CfgBuff[0]=0;
    PCCard_SetCISPointer(0,0);
                                /* アトリビュートメモリ先頭アドレスセット */
    xputs("¥n");
    PCCard_CISTPL_Analysis(CfgBuff,CardName);
                                        /* タプル解析メインルーチン */
    if (CardCheck(CardName)){
        xputs("TEST PIO 16bit PCCard¥n");
        /* カードコンフィグレーション */
        CCR_Address=PCCard_CISTPL_GetCCRaddress();
                                        /* CCRアドレス取得する */
        p=(unsigned char *)PC0ATT;
        *(p+CCR_Address)=CfgBuff[0]&0x3f;
                                        /* インデックス番号書き込み */

        /* カードPIO I/Oアドレス取得 */
        PCCard_IOaadr=CfgBuff[4]<<24|CfgBuff[3]<<16|CfgBuff[2]
<<8|CfgBuff[1];     /* 最初のインデックス番号のI/Oアドレス */
        p=(unsigned char *)PC0IO;
        *(p+PCCard_IOaadr+4)=1; /* 割り込みクリア */
        *(p+PCCard_IOaadr+6)=1; /* 割り込みマスク解除 */

        *GFER0 = (*GFER0&3) | 0x220;
                    /* GP9/GP5立ち下りエッジ検出(GP9割り込み入力) */
        *GFER0 = (*GFER0&3) | 0x220;
                    /* GP9/GP5立ち下がりエッジ検出(GP9割り込み入力) */
        PCCARD_Enable=1;    /* カードイネーブル完了 */

    } else {  /* それ以外のカード */
        xputs("Other 16bit PCCard¥n");
    }
}
```

〔リスト4〕ATAフラッシュカードのセクタダンププログラム

```
/* カードがコンフィグレーション状態の時のみ実行 */
if (PCCARD_Enable) {
    p=(unsigned char *)PC0IO;
    switch(ATA_Fase) {
                                ～中略～

        case 3 :  /* セクタリードコマンド発行 */
            *(p+PCCard_IOaadr+2)=1;             /* 1セクタ */
            *(p+PCCard_IOaadr+3)=LBA&0xff;      /* LBA最下位 */
            *(p+PCCard_IOaadr+4)=(LBA>>8)&0xff; /* LBA中位 */
            *(p+PCCard_IOaadr+5)=(LBA>>16)&0xff;/* LBA上位 */
            *(p+PCCard_IOaadr+6)=0x40; /* マスタドライブ */
            *(p+PCCard_IOaadr+7)=0x20; /* セクタリードコマンド発行 */
            for(i=0;i<0x1000;i++){}    /* ウェイト */
            ATA_Fase++;
            break;
        case 4 :  /* BUSYクリア待ち */
            c=*(p+PCCard_IOaadr+7); /* ステータスリード */
            if ((c&0x80)==0) ATA_Fase++;
            break;
        case 5 :  /* データ準備待ち */
            c=*(p+PCCard_IOaadr+7); /* ステータスリード */
            if (c&8) ATA_Fase++;
            break;
        case 6 :  /* セクタリード */
            w=(unsigned short int *)(PC0IO+PCCard_IOaadr);
            for(i=0;i<256;i++){
                buff[i]=*w;    /* データレジスタ読み出し */
            }
            ATA_Fase=0;         /* アイドルへ */
            LBA++;              /* 次のセクタ */
            break;
        default :               /* 次のセクタリード指示待ち */
            break;
    }
}
```

〔写真2〕PCカードをCFソケットで使うためのアダプタ

DOS環境と異なる点として，DOS環境ではI/O空間がありPIOのレジスタもI/O空間にマッピングされましたが，ARMアーキテクチャにはI/O空間はなく，すべてメモリマップドI/Oになる点です．**図1**のメモリマップで示したように，ソケット0のI/O空間は0x20000000からのアドレスとなります．

そしてもう一つ，I/Oコンフィグレーションのアルゴリズムが異なります．DOS版では，このPIO PCカードでコンフィグレーション可能なアドレスのうち，動作させる環境で未使用のI/Oアドレスをあらかじめ選んでおき（#define定義を書き換える），そのアドレスのみを使うようになっています．

しかしPXA25xでは，何度も説明しているように，ソケット0のI/O空間として，64Mバイトの空間がまるまる確保されています．つまり，I/Oカードはどのアドレスにコンフィグレーションしても，システムのI/Oレジスタとぶつかることはないで，これからコンフィグレーションするI/Oアドレスが，システムですでに使われているI/Oアドレスと重なるかどうかといった判定をする必要はありません．

そこで，カードイネーブラのI/Oコンフィグレーションのアルゴリズムとしては，属性情報を検索して最初に発見したインデックス番号をそのままカードのコンフィグレーションレジスタ（CCR）に書き込むようにしています．

リスト3にPIO PCカードの割り込みカウントプログラムを示します．

● ATAフラッシュカードのセクタダンプ

さらに，参考文献1）のPIO PCカードをもっていない場合の

Column 2

PCカードインターフェース機能を使ったISAバスブリッジについて

● 相互接続可能か？

PCカードはデータバス幅16ビットで，アドレス空間も64Mバイトあります．この機能を使って，データバス16ビット，アドレス空間16MバイトのISAバスへのブリッジはできないものでしょうか？ **表A**に対応させる信号の案を示します．ISAバスではDMA関連の信号もありますが，とりあえずここでは省きます．

まずリセット出力や割り込み入力は，GPIO機能で代用できるでしょう．

$\overline{\text{SMEMR}}$/$\overline{\text{SMEMW}}$は，1Mバイト以下のアドレスに対するメモリリード/ライト信号なので，正確にはA23～A20までがすべて“L”レベルのときにアサートすべきです．

AENはアドレスバスの内容が有効状態のときにアサートされるので，PCE1とPCE2をANDした信号で代用できるでしょう．

BALE信号は，アクセス途中で内容の変わるLA23～LA17をラッチするために使う信号で，タイミング的にはISAバスへのアクセスを開始したタイミングで1クロック～半クロックの間だけ“H”レベルになる信号です．LA23～LA17にアドレスバスA23～A17を代入するなら，アクセス途中でアドレスバスの内容は変化しないので，BALE信号は常時“H”レベルでも問題ないはずです．しかし一部のISAバスボードでは，この信号をバスアクセスの開始信号として使っているものもあるようなので，注意が必要です．

メモリ空間のレジスタ幅を示す$\overline{\text{MEMCS16}}$の処理はやっかいです．PCカードの仕様では，メモリ空間はあくまで16ビット幅であり，I/O空間のようなダイナミックバスサイジング機能はないからです(だからI/O用のIOIS16#しかない)．

そしてクロックも問題です．ISAバスでは約8MHz程度のクロックが必要です．しかもリード/ライトクロックなどと同期している必要もあるでしょう．

● I/Oデバイスのみと仕様を割り切れば

ここでもう一度「ISAバスを使いたい」という要求の本質を考えると，やはりISAバス接続のI/Oデバイスを使いたいのだと思われます．

であるなら，汎用性を高めカードエッジのISAバスのボードをそのまま使うことは考えずに，ISAバスに接続できるタイプのI/OデバイスをPXA25xに直結するというように仕様を割り切ればよいわけです．これなら，メモリ空間は使わないので，$\overline{\text{SMEMR}}$/$\overline{\text{SMEMW}}$や$\overline{\text{MEMCS16}}$，BALE信号も不要です．

残る問題は，クロックをどうするかだけです．デバイスによっては，内部的な動作のクロックとして使うだけで，ISAバス側の信号と同期している必要がない場合もあります．そのようなデバイスなら，単純に8MHz程度のクロックを供給するだけで接続が可能になるでしょう．

〔表A〕ISAバスブリッジを実現する案

ISAバス信号名称	意　味	PXA25x信号名称
CLK	クロック	？
RSTDRV	リセット	GPIOx
SA19～SA0	アドレスバス(1Mバイト)	A19～A0
LA23～LA17	アドレスバス(上位)	A23～A17
SD15～SD0	データバス	D15～D0
$\overline{\text{SMEMR}}$	システムメモリリード	OE？
$\overline{\text{SMEMW}}$	システムメモリライト	PWE？
$\overline{\text{MEMR}}$	メモリリード	OE
$\overline{\text{MEMW}}$	メモリライト	PWE
$\overline{\text{IOR}}$	I/Oリード	PIOR
$\overline{\text{IOW}}$	I/Oライト	PIOW
IOCHRDY	I/Oレディ	PWAIT
$\overline{\text{IOCS16}}$	16ビットI/O	PIOIS16
$\overline{\text{MEMCS16}}$	16ビットメモリ	？
$\overline{\text{AEN}}$	アドレスイネーブル	PCE1 & PCE2
$\overline{\text{SBHE}}$	ハイバイトイネーブル	PCE2
BALE	LA23～LA17ラッチ	常時“H”レベル？
$\overline{\text{IRQx}}$	割り込み信号	GPIOx

動作確認プログラムも作成しました．CFカードでもっとも一般的なカードといえば，フラッシュATAカードだと思われます．そこでフラッシュATAカードを差し込んだら，セクタを先頭からダンプしていくプログラムを用意しました(**リスト4**)．

フラッシュATAカードとは，名前に「ATA」がついているように，カードをコンフィグレーションした後は，そのI/OアドレスにはATAレジスタがマッピングされ，IDEのHDDと同様の制御プログラムで読み書きを行うことができます．ATAレジスタの制御方法についての詳細は，参考文献2)を参照してください．

先ほどのPIO PCカードでは，属性情報に格納されているメーカー名や製品名の文字列が一致するかどうかを検索して，PIO PCカードであるかどうかを判定していました．

ATAカードでは，メーカーや製品が異なっても，ATAという共通仕様に基づいた製品であれば，同一のプログラムで制御することができます．そのため，メーカー名や製品名の文字列でカードを判定するのではなく，ATA仕様に準拠しているかどうかを判定して，カードをコンフィグレーションしています．

*　　　　　*

次回は，本評価ボードとPCIバスを接続するPCIバスブリッジを作成する予定です．ご期待ください．

参考文献

1)『PCカード/メモリカードの徹底研究』，TECH I Vol.14，CQ出版(株)
2)『ATA(IDE)/ATAPIの徹底活用研究』，TECH I Vol.10，CQ出版(株)

やまたけ・いちろう　来栖川電工(有)

音楽配信技術の最新動向

OggVorbisの現状について

―― Ogg Vorbis

第7回（最終回）

岸　哲夫

この連載記事で，OggVorbisを扱うことが可能になったかと思います．エンコードもデコードも，それ自体は簡単です．いかに使いやすい衣(GUI)をかぶせるかで，見栄えも使い勝手も大きく違ってきます．オープンソースで役に立つツールを作成し，普及に役立ててみてください．OggVorbisに関しての音響技術的な話題については機会を改めて，より詳細に行う予定でいます．

● OggVorbis対応ハードウェアの登場

業界のいろいろな力関係によって，OggVorbisはまだメジャーなものになっていないようです．一説によると，米国でいう「サブマリン特許」の類がOggVorbisに含まれていないかどうかで，皆が二の足を踏んでいて，OggVorbis対応のハード機器を出すのが遅れているようです．

iRiver社は去年から，既存の商品にもファームウェアのバージョンアップで「対応する」と言い切ったまま1年近くがすぎています．

しかし最近，OggVorbis対応のハードウェアが出るとの話を聞きました．それはRio Audioから発売予定のKarmaというプレーヤです(**図1**)．OggVorbisやFlacに対応し，さらにEthernetポートも付いています．もちろんHDD内蔵プレーヤです．

先鞭を切った企業が出たことで，対応するメーカーも続々と増えるでしょう．またiRiver社も，iHP-120というiPodタイプの商品を10月上旬に発売する予定です(**図2**)．録音再生可能なポータブルデジタルデバイスとのふれこみで，OggVorbisにも対応しているようです．なぜかまだWindowsMedia9の可変ビットレートには対応できないようですが．

● 音楽配信と利権問題について

今後個人で利用できるインターネット回線の容量が増大し，CD1枚分程度の情報なら瞬時に転送可能になってきたとき，音楽家やプロダクションは，聞き手に向かって作品をデータ配信で受け渡すことも視野に入れるかもしれません．もっともジャケットも含めて作品であるという観点から，そのような方式は

〔図1〕Rio Audioのサイト(http://www.digitalnetworksna.com/rioaudio/default.asp?cat=35)

〔図2〕iRiverのサイト(http://www.iriverjapan.com/product.php)

一般化しないという声もあります．しかし，「レコードはビジュアル的刺激も聞き手に与えてくれた」，「CDでは満足できない」という声は見事に無視され，10年ほど前にはレコード針が入手不可能という事態にまで発展したことを記憶にとどめていると思います．しかし，その後のLo-Fi，スクラッチなどのクラブ・DJ文化の影響で新作のレコードを出すミュージシャンも増えてはいます．

著作権管理をきちんと行って音楽配信することができるなら，著作権を管理する組織や業種としてのレコード業には未来はないでしょう．逆にそのあたりがしっかりしていないために，音楽配信自体がグレーゾーンのように扱われてしまうのです．

米国では，全米レコード協会（RIAA）がMP3を提訴する事態にまで発展しました．問題は落ち着きましたが，この争いは，その後実質的にMP3が商用で利用できなくなったことと無関係ではないと思います．1989年にドイツにおいてフラウンホーファー研究所にMP3の特許が与えられました．MP3はその数年後，国際標準化機構（ISO）に提出され，MPEG-1仕様に統合されたのです．

最新ではないのですが現在も配布されているRed Hat Linux 8.0では，パッケージに含まれる音楽再生ソフト「xmms」でMP3を使用することができないようにコンパイルされています．もちろん自分でソースをダウンロードしてコンパイルすれば問題はありませんが．

ネット上のCDショップでは視聴用にMP3を置くのが一般的だったようですが，それも商用利用禁止によって混乱しています．WMAだけで置いてあったりすると，Macユーザーなどは困ってしまうでしょう．

そんな状況なので，ユーザーはマルチプラットホームでライセンスフリーな圧縮ソフトを求めます．そこでOggVorbisというフォーマットが脚光を浴びることになったのです．

普及しない理由はOggVorbisにはありません．ユーザーや音楽家の意向を無視して，旧弊にしがみつく音楽業界の問題です．音楽配信を行おうとした企業が問題を起こして自滅したのは，象徴的な出来事でした．

● 音楽配信の今後は？

ビデオオンデマンドの実用化さえいわれている昨今，音楽でできないはずはないのですが，いろいろな波紋を引き起こすようです．ビートルズの過去の作品の著作権管理だけが業務であるアップルレコードが，アップルコンピュータが行っている音楽配信に「紛らわしいので訴訟する」と屁理屈のような行動をとるのも，音楽配信という概念が業界再編に直結するからにほかなりません．またせっかく音楽配信を行っているのにWindows以外では視聴不可能な規格もあります．これでは普及の足を引っ張るだけです．PCで音楽を聴きたい人達は結局モラルを捨ててP2Pファイル交換に走ってしまうかもしれません．

回線インフラが増強されると，音楽は圧縮されずに，またはFlacなどの可逆圧縮形式で配信されるようになるかもしれません．そのときにはますます正しい著作権管理が重要になってくるでしょう．表現者の側の理想は音楽や映像をユーザーが視聴するごとに課金されることだと思います．そうすればオークションサイトで音楽をコピーして売ったり，中古音源ブローカーは壊滅するでしょう．とはいえインターネットまたはそれに類するものがないと，音楽や映像を視聴できなくなるのは寂しいことですが．

〔図3〕フレッツ・スクウェアのサイト
（http://flets.com/square/index.html）

また，音楽配信技術はPCだけのものではありません．PHSや携帯電話，PDAさえもその技術の恩恵を受けることになります．外部メモリ付き電話端末の一般化は，インフラの整備となって，状況が変わっていくはずです．米RealNetworksが英Vodafoneと提携する事例でわかるように，これから大きな変化が起きると思います．10代が消費する音楽をPCで配信しようとしても，（ターゲットたる10代が）クレジットカードをもっていなかったりPCをもっていなかったりして普及の阻害となっていました．しかし，この方法なら携帯電話の料金とともに配信料金を課金することができますし，PCが不要であることでこれから期待できる分野だと思います．

駅や繁華街周辺ではPDAに無線LANカードを差込みインターネットに接続しているユーザーも多数います．そのようなユーザーも，無線LAN接続料金とともに課金可能なシステムを利用できるので普及しやすいのではないかと思います．

映像も含めたコンテンツ配信は，ネットのプロバイダが行い課金するフレッツ・スクウェア（**図3**）のようなビジネスモデルが普及していくことでしょう．しかし内容がお粗末だったり，

料金が高価だったりすると，「幻のニューメディア」(?!)キャプテンシステムの轍をふむことになりかねません．

● 可逆圧縮コーデック「Flac」について

将来一般のユーザーが使用する回線インフラがますます増強されると，可逆圧縮コーデックによって配信されることになるかもしれません．ここでは，その一つであるFlacについて簡単に説明します．

Flacとは，Free Lossless Audio Codecの略です．いくつか可逆圧縮コーデックがありますが，オープンソースであることや，その機能の高さが知られています．

MP3やOggVorbisは不可逆音声圧縮を行います．すると当然のことながら，一度圧縮したら元のデータと100％同じデータには戻りません．Flacの場合，元のデータと100％同じものに復元できます．ちょうどデータファイルの圧縮や展開と同じです．

Flacは対象の音楽にもよりますが，最大で40％程度の圧縮ができます．どちらかといえばテクノ系などずっと音が鳴り続けている音楽より，アコースティック系の音楽のほうが圧縮率が高くなります．

なお2003年の1月に，OggVorbisを開発する団体であるXiph.orgに可逆圧縮コーデックプロジェクトのFlacが参加し，Oggプロジェクトの一つになりました．まだ旧Flac公式サイトも新・公式サイトも存在しています．xiph公式サイトからFlacのサイトにリンクが張られています(**図4**，**図5**)．

● Flacで圧縮して保管するメリット

過去にMP3でリッピングをして保存していた音楽をOgg Vorbisで保存しなおしたいと思っても，再度リッピングしなおさなければなりません．MP3形式をWAV形式に戻して，それをvorbis形式にすることで，音質はコーデック変換のたびに低下してしまいます．不可逆の圧縮をした場合は圧縮時に音を間引いているため，WAV形式に戻しても，リッピング時点のWAV形式データと同一にならないからです．

そこでFlac形式で元データを保管していれば，Flac→WAV→MP3と変換してもFlac→WAV→Vorbisと変換しても理論上は音質が変わらなくなります．

これで，手持ちのCDを全部リッピングして自宅内で音楽サーバを作ることも可能になります．筆者も貴重なCDを紛失しないように，Flacに変換後DVDに焼いて保管しています．CD10～15枚ほどの内容を1枚のDVDに格納できます．

ちなみにFlacのデータフォーマットは思ったほど複雑ではありません．公式サイトの情報によると，**表1**のようになっています．

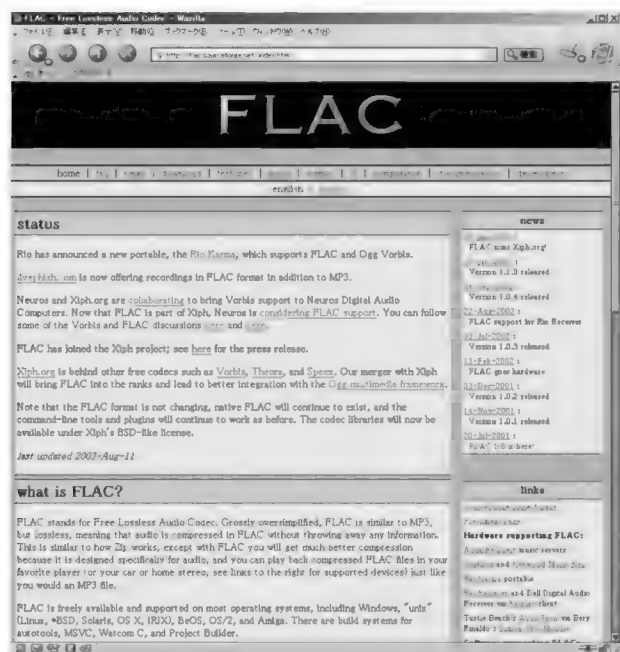
● 映像配信について――「MediaWiz」を知っていますか？

この連載は今月で終了しますが，次は映像配信について書いてみたいと考えています．映像配信に関することが縦糸ならば，たとえばMediaWizを使い倒してみようというのが横糸になります．各種コーデックの説明も含めて，濃い記事にするつもりですので期待してください．

ところでMediaWizは，最初からモノが少ないのかもしれませんが，店に在庫がなくなるほど売れているようです．この商品はそれだけで小規模なストリーミング配信サーバや，ビデオオンデマンドができてしまうものです．もちろんOggVorbisを再生することができます．日本ではバーテックスリンクが販売しています．某匿名巨大掲示板にも専用スレッドが立っていました．

このMediaWizは，PCに保存してある音声や動画を自らデ

〔図4〕Flac公式サイト(http://xiph.org/)

〔図5〕Xiph公式サイト
(http://flac.sourceforge.net/index.html)

〔表1〕**Flacのデータフォーマット**(`http://flac.sourceforge.net/format.html`より)

ストリーム関連データ		データ長
識別子	この内容は必ず"fLaC"である	32ビット長
METADATA_BLOCK	たとえばサンプリングレート値のような全体に関係する情報，チャネルの数などがSTREAMINFOメタデータブロックに記述されている	任意
METADATA_BLOCK	データブロックが繰り返されることがある	任意
オーディオデータ	一つ以上の場合もある	任意
METADATA_BLOCK		
ヘッダ部	METADATA_BLOCKのタイプやサイズがセットされている	任意
データ部		任意
METADATA_BLOCK　ヘッダ部		
メタデータブロックフラグ	このブロックがオーディオブロックの前の最後のメタデータブロックである場合「1」，そうでなければ「0」がセットされている	1ビット
ブロックタイプ	0：STREAMINFO 1：PADDING 2：APPLICATION 3：SEEKTABLE 4：VORBIS_COMMENT 5：CUESHEET 6～127：予約済み	7ビット
METADATA_BLOCKのバイト長	これ以降のメタデータの長さ(単位はバイト)	24ビット
METADATA_BLOCK　データ部		
METADATA_BLOCK_STREAMINFO METADATA_BLOCK_PADDING METADATA_BLOCK_APPLICATION METADATA_BLOCK_SEEKTABLE METADATA_BLOCK_VORBIS_COMMENT METADATA_BLOCK_CUESHEET	ヘッダ部の「ブロックタイプ」に依存する	任意
METADATA_BLOCK_STREAMINFO		
最小ブロックサイズ	ストリーム中で最小のブロックサイズの値	16ビット
最大ブロックサイズ	ストリーム中で最大のブロックサイズの値	16ビット
最小フレームサイズ	ストリーム中で最小のフレームサイズの値，ただし明確でないときは0	24ビット
最大フレームサイズ	ストリーム中で最大のフレームサイズの値，ただし明確でないときは0	24ビット
サンプリングレート	Hzでサンプリングレートの値を表す，20ビットだが最大値は1048570Hzである．0はあり得ない	20ビット
チャネルの値	ただし，実際のチャネル値から1を減じた値がセットされている．Flacは1～8チャネルに対応している	3ビット
サンプリングビットの値	ただし，実際のサンプリングビット値から1を減じた値がセットされている．5ビットあるので32ビットまで対応できるはずだが，現在は4～24ビットまで対応している	5ビット
全体のサンプリングの値	モノラルのオーディオデータで，サンプリングレートが1秒で44.1kHzならばサンプリング値は44100である． もしオーディオデータ全体がその値ならば，ステレオの場合サンプリング値を固定するなら，サンプリングレートを半分にしなければならない． そのデータがセットされている．	36ビット
ディジタル署名	MD5を使ったディジタル署名データがセットされている	128ビット
METADATA_BLOCK_PADDING		
桁合わせ	0値が8の倍数ビット分セットされている	8の倍数ビット
METADATA_BLOCK_APPLICATION		
作成したアプリケーションの識別値	`http://flac.sourceforge.net/id.html`のページでアプリケーションIDを登録することができる	32ビット
アプリケーション固有のデータ	作成したアプリケーションはここに独自のデータを設定できる8の倍数ビット分セット可能	8の倍数ビット
METADATA_BLOCK_SEEKTABLE		
検索インデックスデータブロック	一つまたはそれ以上のブロックが指定される	
METADATA_BLOCK_VORBIS_COMMENT		
コメント	`http://www.xiph.org/ogg/vorbis/doc/v-comment.html` このページにあるようなvorbisコメントパケットの内容がセットされている	任意
METADATA_BLOCK_CUESHEET		
メディアのカタログ情報	ASCII印刷可能な文字0x20～0x7eの中で，ランダムにカタログ番号を作る． 一般に，メディアカタログ番号は長さ0～128バイトのはずである． CD-DAの場合は，128バイト．内訳は13桁の数字で，残りはNULL値	128×8ビット

〔表1〕**Flacのデータフォーマット**(http://flac.sourceforge.net/format.htmlより)(つづき)

METADATA_BLOCK_CUESHEET		
リードイン部分のサンプリング数	このフィールドは，CD-DAキューシートでのみ意味をもっている． 他の場合は0である． CD-DAについては，リードイン部が目次が格納されているTRACK 00エリアである．つまりメディアの最初のサンプリングデータからTRACK 01の先頭インデックスポイントの先頭データまでのサンプルの数である．レッドブックによれば，リードイン部は無音のはず．また，リッピングソフトウェアは通常それを格納しない．そしてリードイン部は少なくとも2秒以上である． これらの理由のため，先頭トラックの絶対的な位置を計算することができるように，リードイン部の長さはここに格納される．ここに格納されたリードイン部が先頭トラックの先頭インデックスまでのサンプルの数であることに注意すること	64ビット
フラグ	対象がCDの場合は1，その他の場合は0	1ビット
予約済み領域	NULL値	7＋258×8ビット
トラック数	CDのトラック数	8ビット
キューシート情報	くわしくはhttp://flac.sourceforge.net/format.htmlを参照	

〔図6〕**Uzuのサイト**(http://www.geocities.co.jp/SiliconValley-Cupertino/2647/)

コードするので，素材のサーバとしてのPCにかかる負荷はかなり軽減されます．しかし同梱されているソフトMediaWiz Serverに，かなり問題があります．ユーザーはこの使用感に耐えかね，なんとフリーの互換ソフトを作ってしまいました．Uzuというソフトです(**図6**)．もしMediaWizを購入したなら，Uzuを使ってみるとよいでしょう．

● コンテンツ配信とオープンソース

映像も音楽も含めたコンテンツ配信を行う際に注意しなくてはならないことは，まず著作権です．コンテンツの著作権は当然のことですが，配信技術のライセンスのことも考えなくてはなりません．

少し前までは，JASRACはコンテンツ配信を実質的に認めていませんでしたが，現在はネットワーク課というセクションもできて，スムーズに事を運んでいるようです．外から見て異常(?!)な集団だったJASRACにも，内部から改革する力が出てきたようです．

配信技術のライセンスも重要です．MP3の例を見てもわかるように，配信手段や方法にライセンスが発生した場合，配信者はそのライセンス料金も支払う事態になってしまうでしょう．そこで，オープンソースであるOggVorbisが重要です．以前はライセンスの問題でGIFが実質的に使用できなくなりました．今はJPEGももめているようです．映像のほうでもMPEG-4規格に何か起きたら困るということで，オープンソースプロジェクトが進行しています．

サブマリン特許やビジネスモデル特許も行き過ぎた運用をすれば，自分の首を絞めてしまうと思います．目先の利益を出すことだけが絶対というシステムを信仰する文化も尊重しなくてはならないと思うので，とやかくケチをつけることではありません．そういうモノにかかわらなくてもすむように，オープンソースの普及に期待したいものです．

どこでもWindowsを使うことを前提にされ，映像も音楽もWindowsでしか動作しないコーデックを押し付けられるのは勘弁してほしいものです(WMA9はたしかに一般的なMP3プレーヤより高性能だとは思うが)．

＊　　＊

以前紹介したミュージシャンのピアツーピア配信実験や，個人で簡単にできるストリーミングサーバなどが，ますますネットワークを楽しくしていくことでしょう．本連載はひとまず終了します．

機会があったら映像配信技術や，OpenAudioLicenceの現状，オープンソースで映像・音楽を作り配信する手段なども記事にしたいと思います．

きし・てつお

x86CPUだけでもマスタしたい

開発技術者のためのアセンブラ入門

第22回 SIMD命令 (1) MMX命令

大貫広幸

今回と次回とで，x86系CPUがもつSIMD命令について，その概要を説明する予定です．x86系CPUがもつSIMD命令は，MMXやSSE，SSE2といった名称で呼ばれています．

SIMD命令とは？

これまで説明してきたx86系CPUは，一つの命令で一つのデータを処理してきました．たとえば，

```
ADD  AX, DX
```

という16ビット加算命令は，レジスタAXにレジスタDXの値を16ビット長で加算するという，一つの加算処理を一つの命令で行っています．このように，一つの命令が一つのデータを処理する命令のことを“Single Instruction Single Data”，略して「SISD」と呼びます．

これに対して，一つの命令が複数のデータを処理する命令のことを“Single Instruction Multiple Data”，略して「SIMD」と呼びます．たとえば，

```
PADDW  MM0, MM1
```

というx86系CPUのSIMD命令は，レジスタMM0上の四つの16ビット値に，レジスタMM1上の四つの16ビット値を，それぞれの要素ごとに16ビット加算するという，四つの加算処理を一つの命令で実行するものです．

このように，SISD命令では複数ステップ必要な演算プログラムも，SIMD命令では1命令で実行できるため，配列やベクトルに対して行われる，同じ演算を繰り返すようなプログラムでは，SISD命令に比べSIMD命令を使用したほうが，実行速度の面で断然有利だといえます(**図1**)．

● Pentiumプロセッサと SIMD命令

現在使われている，Pentiumプロセッサは，初期のものを除き，このSIMD型の命令を備えています．Pentiumプロセッサで実行できるSIMD命令は，整数演算を行う「MMX命令」，単精度浮動小数点演算を行う「ストリーミングSIMD拡張(SSE)命令」，そして倍精度浮動小数点演算を行う「ストリーミングSIMD拡張2(SSE2)命令」の3種類です．

これらSIMD命令は，CPUの発売時期により使用できる命令セットが異なります(**図2**)．MMX命令がいちばん古く，MMX Pentium以降のCPUで使用できます．そして，SSE命令はPentium III以降，SSE2命令が最新のPentium4ということになります．自分のCPUが，どの種類のSIMD命令を実行可能かは，CPUID命令で知ることができます(**コラム1**参照)．

● MMX，SSE/SSE2で扱うデータ

x86系CPUのSIMD命令で処理するデータは，MMXが64ビット長，SSEおよびSSE2が128ビット長の固定長となっています．この固定長のビット内に，同じ型のデータを複数個，

〔図1〕SISD命令とSIMD命令

次の4個のワード値の加算でSISD命令とSIMD命令の違いを見る

SISD命令

・C0←A0＋B0を計算
・C1←A1＋B1を計算
・C2←A2＋B2を計算
・C3←A3＋B3を計算

SISD命令では，C0～C3を求めるのに4ステップが必要

SIMD命令

・C0←C1＋B0
　～
　C3←C3＋B3
　の四つの計算を一つの命令で行う

SIMD命令では，C0～C3を求めるのに1ステップで済む

〔図2〕x86系CPUと使用できるSIMD命令の種類

CPUID命令で CPUがサポートする機能を調べる

本文でも述べているように，x86系CPUは発表時期によりサポートされている命令が違っています．このサポートされている命令を調べるには，CPUID命令を使用します．

CPUID命令は，Pentium以降のプロセッサならすべて実行することができます．ただし，古いタイプの486や386といったCPUには，CPUID命令自体がないので，CPUID命令は使用できません．

CPUID命令でCPUがサポートする機能を調べる場合，まずレジスタEAXに0を設定し，CPUID命令を実行します．その結果，レジスタEAXが1以上の値になっていれば，サポート機能の取得が可能です．もし，レジスタEAXが0だった場合は，そのCPUはサポート機能の取得ができない，つまりサポート機能なしと言うことになります．

次に，レジスタEAXを1にしてCPUID命令を実行します．するとCPUID命令は，レジスタEDXに**表A**のような，CPUがサポートしている機能の一覧を示したビットを返してきます．

このビットを見て，自分が使用したい機能がCPUにあるか否かを判断します．ビットが1ならCPUにその機能のサポートがあり，ビットが0ならCPUにその機能のサポートなしです．

たとえば，自分が使用しているCPUに，SIMD命令のサポートがあるか否かは，**リストA**のようなプログラムで調べることができます．

〔表A〕EAX=1でCPUID命令がEDXに返してくる内容

ビット	名　称	内　　容
31～28	予約	
27	SLFSNP	セルフスヌープ
26	SSE2	ストリーミングSIMD拡張命令2
25	SSE	ストリーミングSIMD拡張命令
24	FXSR	FXSAVE/FXRSTOR
23	MMX	MMXテクノロジ
22	ACPI	プロセッサパフォーム監視
21	DTES	デバッグトレース/イベントモニタ
20	予約	
19	CLFSH	CFLUSH 命令
18	PSN	プロセッサシリアル番号
17	PSE	ページサイズ拡張
16	PAT	ページ属性テーブル
15	CMOV	条件付き転送および比較命令
14	MCA	マシンチェックアーキテクチャ
13	PGE	PTE グローバルビット
12	MTRR	メモリタイプ範囲レジスタ
11	SEP	SYSENTER および SYSEXIT
10	予約	
9	APIC	オンチップ APIC
8	CX8	CMPXCHG8B 命令
7	MCE	マシンチェック例外
6	PAE	物理アドレス拡張
5	MSR	RDMSR および WRMSR サポート
4	TSC	タイムスタンプカウンタ
3	PSE	ページサイズ拡張
2	DE	デバッグ拡張
1	VME	仮想8086 モード強化
0	FPU	オンチップ FPU

〔リストA〕CPUID命令で使用可能なSIMD命令を調べるプログラム例

```
        mov     eax,0
        cpuid
        cmp     eax,1
        jb      noSupportCPUIDeax1
        mov     eax,1
        cpuid
        test    edx,04000000h
        jz      noSupportSSE2

        SSE2命令が使用できる．
        この場合，SSE命令，MMX命令も使用可能

noSupportSSE2:
        test    edx,02000000h
        jz      noSupportSSE

        SSE命令が使用できる．
        この場合，MMX命令も使用可能

noSupportSSE:
        test    edx,00800000h
        jz      noSupportMMX

        MMX命令のみ使用できる

noSupportMMX:

        SIMD命令は使用できない

noSupportCPUIDeax1:
        CPUID命令でサポートに関する情報は取得できない．
        この場合，SIMD命令は使用できないと判断する
```

配列として配置することになります．この状態を「パック」といいます（**図3**）．メモリアクセスおよび演算は，このパックされた状態で行われます．

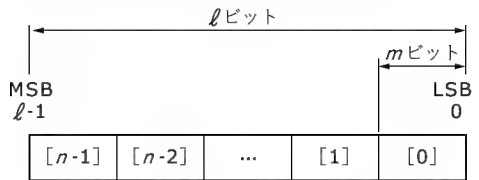
機能的にはMMXが，整数のみの64ビット長のパックドバイト整数，パックドワード整数，パックドダブルワード整数，クワッドワード整数の値を扱うものです．SSEは，128ビット長のパックド単精度浮動小数点数が扱えます．SSE2ではさらにパックド倍精度浮動小数点数に加え，128ビット長のパックドバイト整数，パックドワード整数，パックドダブルワード整数，パックドクワッドワード整数，ダブルクワッドワード整数の値を扱うことができるようになっています．

SIMD命令が処理するデータ（要素）の形式は，CPU本体のSISD命令と同じです．つまり，整数値は2進数で，符号付きは2の補数で表されます．浮動小数点値は，IEEE規格754の単精度と倍精度の2進浮動小数点形式となっています．

図4は，MMX，SSE/SSE2で扱えるデータの種類を示したものです．

〔図3〕SIMD命令が扱うパックされたデータ

ℓビット
mビット
MSB ℓ-1
LSB 0

[n-1]	[n-2]	…	[1]	[0]

・上記のデータはℓビット中に同じ型のm ビットの値が[0]〜[n-1] まで連続してn 個，つまりℓ =m ×n の関係にある
・SIMD命令で使用されるレジスタはℓ ビットの長さをもつ
・メモリアクセスはℓ ビットの単位で行われる

〔図4〕SIMD命令が扱うデータ

・整数は2進数で符号付きと符号なしがある．符号付きの負数は2の補数で表される
・浮動小数点(単精度と倍精度)はIEEE規格754の形式

〔図5〕MMXレジスタ

MMXレジスタとFPUのレジスタスタックは，物理的に同じレジスタを共有している

MMX命令の使用方法

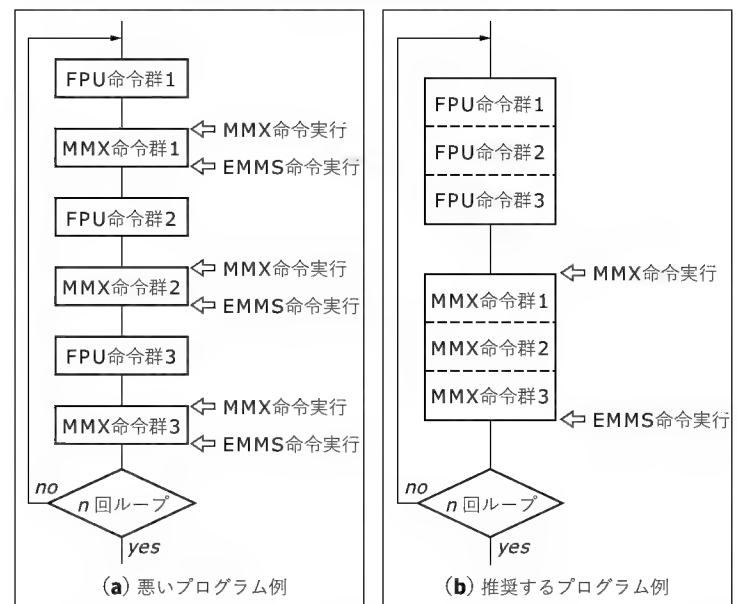
初めにも述べたようにMMX命令は，パックされた2進整数の演算を行うSIMD命令です．メモリ上，あるいはレジスタ上のデータは64ビット固定長なので，1パックあたりパックドバイト整数なら8個，パックドワード整数なら4個，パックドダブルワード整数なら2個，クワッドワード整数なら1個のデータを一つの命令で処理することができます．

● MMX命令が使用するレジスタとプログラミング上の注意

MMX命令では，MMXレジスタと呼ばれるレジスタにパックされたデータをロードし，演算します．

MMXレジスタは，一つ64ビット長のレジスタが全部で8本あります(**図5**)．このMMXレジスタは，FPUのレジスタスタックと共有されています．そのため，この共有レジスタはFPU命令実行時には浮動小数点値を格納し，MMX命令実行時にはパックされた整数値を格納することになります．

〔図6〕FPU命令とMMX命令のプログラム記述上の注意

(**a**) 悪いプログラム例　(**b**) 推奨するプログラム例

・上図の ⇦ の所でFPUのレジスタスタックとMMXレジスタの切り替えが行われる．レジスタの切り替えには時間がかかるので，実行速度の面からできるだけこの切り替えを行わないほうがよい．
・FPU命令とMMX命令に厳密な前後関係がなければ，(a) のプログラムは (b) のようにプログラムしたほうが実行速度が速くなる．

FPUのレジスタスタックからMMXレジスタへの切り替えは，FPUのレジスタスタックがすべて空のときにMMX命令を実行することで自動的に行われます．逆に，MMXレジスタからFPUのレジスタスタックへの切り替えは“EMMS”という命令で行います．このレジスタの切り替えには時間がかかるため，頻繁にレジスタ切り替えを行うことは，プログラムの実行速度の低下をまねくことになります．

このようなことからプログラム上，FPU命令とMMX命令は混在しないようにする必要があります．つまり，FPU命令はFPU命令でまとめて記述し，MMX命令はMMX命令でまとめて記述するようにします(**図6**，前頁)．

● アセンブラMASMおよびgasでのMMX命令の使用

この連載で使用しているMASM(Ver 6.14)で，MMX命令を使用する場合，ソースファイルの頭で行っているCPU指定のディレクティブをPentiumプロセッサを示す「.586」にします．そして次の行にMMX命令の使用をアセンブラに指示するディレクティブ「.MMX」を記述します．

つまり，アセンブラのソースファイルの頭で

```
.586
.MMX
```

の2行を記述することで，MMX命令が使用できるようになります．また，この連載で使用しているLinuxのアセンブラgas(Ver 2.10.90)は，MMX対応なので，そのままの状態でMMX命令が使用できます．

● MMX命令とオペランド

MMX命令には，**表1**のような命令があります．MASMもgasもこの**表1**のインストラクション名が，そのままニモニックとして使用されています．ただし，gasのMMXのニモニックには，型を示す文字は付きません．

EMMS命令には，オペランドがありませんが，それ以外のMMX命令は二つのオペランドをもちます．MOVD命令とシフト命令以外のMMX命令はすべて，64ビットデータを示す転送先，転送元の二つのオペランドをもちます．

MOVD命令とシフト命令，MOVQ命令以外のMMX命令は，転送先(DEST)のオペランドはMMXレジスタと決められています．そして，転送元(SOU)のオペランドにはMMXレジスタあるいはメモリ上にあるクワッドワード(64ビット長)のデータを指定します．もちろん，メモリ上にあるクワッドワードのデータは，MMX命令で指定する型に適した，パックされた整数データである必要があります．

演算では，演算をopとすると「DEST←DEST op SOU」と演算することになります．

MOVQ命令は，転送先および転送元の両方のオペランドにMMXレジスタあるいはメモリ上にあるクワッドワード(64ビット長)のデータが指定できます．ただし，メモリ-メモリ間の転送は指定できません．MOVD命令は，ダブルワード値の転送命令なので，「MMXレジスタ←ダブルワード値」の転送なら，転送先のオペランドにはMMXレジスタ，転送元にはダブルワード値が格納されたCPUの32ビット汎用レジスタか，メモリ上のダブルワード値を示すようにします．

逆に「ダブルワード値←MMXレジスタ」の転送なら，転送元のオペランドとしてMMXレジスタを指定し，転送先のオペラ

〔表1〕MMX命令の一覧

分類	インストラクション名	動作
データ転送	MOVD	Move Doubleword ● 汎用レジスタあるいはメモリ上のダブルワード値をゼロ拡張した，64ビット値をMMXレジスタに格納する ● MMXレジスタ上の下位32ビットの値を，汎用レジスタあるいはメモリにダブルワード値として転送する
	MOVQ	Move Quadword ● MMXレジスタあるいはメモリ上のクワッドワード値をMMXレジスタに格納する ● MMXレジスタ上のクワッドワードの値をMMXレジスタあるいはメモリに転送する
変換命令	PACKSSWB PACKSSDW	Pack with Signed Saturation ● SOUを上位，DESTを下位とするn個の要素のパックド符号付き整数を，飽和処理しビット数が半分のn個の要素のパックド符号付き整数に変換する ① PACKSSWB命令は，8要素のパックド符号付きワード整数を8要素のパックド符号付きバイト整数に変換する ② PACKSSDW命令は，4要素のパックド符号付きダブルワード整数を4要素のパックド符号付きワード整数に変換する
	PACKUSWB	Pack with Unsigned Saturation ● SOUを上位，DESTを下位とする8要素のパックド符号付きワード整数を飽和処理し，8要素のパックド符号なしバイト整数に変換する
	PUNPCKHBW PUNPCKHWD PUNPCKHDQ	Unpack High Packed Data ● SOUの上位32ビットとDESTの上位32ビットの各要素をアンパックし，インタリーブしてDESTに格納する ① PUNPCKHBW命令は，パックドバイト整数 ② PUNPCKHWD命令は，パックドワード整数 ③ PUNPCKHDQ命令は，パックドダブルワード整数
	PUNPCKLBW PUNPCKLWD PUNPCKLDQ	Unpack Low Packed Data ● SOUの下位32ビットとDESTの下位32ビットの各要素をアンパックし，インタリーブしてDESTに格納する ① PUNPCKLBW命令は，パックドバイト整数 ② PUNPCKLWD命令は，パックドワード整数 ③ PUNPCKLDQ命令は，パックドダブルワード整数
パックド算術命令	PADDB PADDW PADDD	Packed Add ● パックド整数によるDEST←DEST+SOUの各要素ごとのラップアラウンド加算，符号付き，符号なしの両方で使用できる ① PADDB命令が，パックドバイト整数のラップアラウンド加算

〔表1〕MMX命令の一覧（つづき）

分類	インストラクション名	動作
パックド算術命令	PADDB PADDW PADDD	② PADDW命令が，パックドワード整数のラップアラウンド加算 ③ PADDD命令が，パックドダブルワード整数のラップアラウンド加算
	PADDSB PADDSW	Packed Add with Saturation ●パックド符号付き整数による DEST←DEST+SOU の各要素ごとの飽和加算 ① PADDSB命令が，パックド符号付きバイト整数の飽和加算 ② PADDSW命令が，パックド符号付きワード整数の飽和加算
	PADDUSB PADDUSW	Packed Add Unsigned with Saturation ●パックド符号なし整数による DEST←DEST+SOU の各要素ごとの飽和加算 ① PADDUSB命令が，パックド符号なしバイト整数の飽和加算 ② PADDUSW命令が，パックド符号なしワード整数の飽和加算
	PSUBB PSUBW PSUBD	Packed Subtract ●パックド整数による DEST←DEST-SOU の各要素ごとのラップアラウンド減算．符号付き，符号なしの両方で使用できる ① PSUBB命令が，パックドバイト整数のラップアラウンド減算 ② PSUBW命令が，パックドワード整数のラップアラウンド減算 ③ PSUBD命令が，パックドダブルワード整数のラップアラウンド減算
	PSUBSB PSUBSW	Packed Subtract with Saturation ●パックド符号付き整数による DEST←DEST-SOU の各要素ごとの飽和減算 ① PSUBSB命令が，パックド符号付きバイト整数の飽和減算 ② PSUBSW命令が，パックド符号付きワード整数の飽和減算
	PSUBUSB PSUBUSW	Packed Subtract Unsigned with Saturation ●パックド符号なし整数による DEST←DEST-SOU の各要素ごとの飽和減算 ① PSUBUSB命令が，パックド符号なしバイト整数の飽和減算 ② PSUBUSW命令が，パックド符号なしワード整数の飽和減算
	PMULHW	Packed Multiply High signed ●パックド符号付きワード整数による DEST × SOU の要素ごとの乗算を行い，各要素の積の上位16ビットのみをDESTに格納する
	PMULLW	Packed Multiply Low signed ●パックド符号付きワード整数による DEST × SOU の要素ごとの乗算を行い，各要素の積の下位16ビットのみをDESTに格納する
	PMADDWD	Packed Multiply and Add ●最初にパックド符号付きワード整数による DEST × SOU の要素ごとの乗算を行い，上位下位で隣接する積（ダブルワード値）の加算を行う たとえば，DEST=[D C B A], SOU=[d c b a]として， PMADDWD DEST，SOU を実行すると，DEST=[(D × d)+(C × c) (B × b)+(A × a)] が得られる
比較命令	PCMPEQB PCMPEQW PCMPEQD	Packed Compare for Equal ●パックド整数をDESTとSOUの各要素ごとに比較し，その結果等しければ，DESTのその要素のビットをすべて1にセットし，等しくなければ，DESTのその要素のビットをすべて0にセットする ① PCMPEQB命令が，パックドバイト整数の比較 ② PCMPEQW命令が，パックドワード整数の比較 ③ PCMPEQD命令が，パックドダブルワード整数の比較
	PCMPGTB PCMPGTW PCMPGTD	Packed Compare for Greater Then ●パックド整数をDESTとSOUの各要素ごとに符号付きで比較し，その結果DEST>SOUとなる要素があれば，DESTのその要素のビットをすべて1にセット，そうでなければ，DESTのその要素のビットをすべて0にセットする ① PCMPGTB命令が，パックド符号付きバイト整数の比較 ② PCMPGTW命令が，パックド符号付きワード整数の比較 ③ PCMPGTD命令が，パックド符号付きダブルワード整数の比較
論理演算命令	PAND	Bitwise Logical AND（論理積） ●ビット単位で DEST←DEST and SOU の論理演算を行う
	PANDN	Bitwise Logical AND NOT（否定論理積） ●ビット単位で DEST←(not DEST) and SOU の論理演算を行う
	POR	Bitwise Logical OR（論理和） ●ビット単位で DEST←DEST or SOU の論理演算を行う
	PXOR	Bitwise Logical Exclusive OR（排他的論理和） ●ビット単位で DEST←DEST xor SOU の論理演算を行う
シフト命令	PSLLW PSLLD PSLLQ	Packed Shift Left Logical （論理左シフト） ●各要素を指定ビット数分，LSBにゼロを入れながら左にシフトする ① PSLLW命令は，パックドワード整数の論理左シフト ② PSLLD命令は，パックドダブルワード整数の論理左シフト ③ PSLLQ命令は，クワッドワード整数の論理左シフト
	PSRLW PSRLD PSRLQ	Packed Shift Right Logical（論理右シフト） ●各要素を指定ビット数分，MSBにゼロを入れながら右にシフトする ① PSRLW命令は，パックドワード整数の論理右シフト ② PSRLD命令は，パックドダブルワード整数の論理右シフト ③ PSRLQ命令は，クワッドワード整数の論理右シフト
	PSRAW PSRAD	Packed Shift Right Arithmetic（算術右シフト） ●各要素を指定ビット数分，MSBの符号を変えずに右にシフトする ① PSRAW命令は，パックドワード整数の算術右シフト ② PSRAD命令は，パックドダブルワード整数の算術右シフト
ステート管理命令	EMMS	Empty MMX State ● MMXレジスタとして使われていたFPUのレジスタスタックを，すべて空の状態にし，x87 FPU命令が使用できるようにする

注：表中のDESTはdestination（先），SOUはsource（元）を示す　●表中のMMX命令は，すべてフラグ（EFLAGS）を変化させない

ンドにダブルワード値をストアするCPUの32ビット汎用レジスタか，メモリ上の32ビット領域を指定します．

シフト命令では，二つのオペランドでシフト対象とシフトするビット数を指定します．まず，シフト対象はパックされた整数値で，転送先（DEST）のオペランドとして指定します．そして，カウントオペランドでシフトするビット数を指定します．シフト対象の転送先は，MMXレジスタのみ指定可能です．カウントは符号なし整数で，MMXレジスタあるいはメモリ上のクワッドワード値，8ビットのイミディエイト値で指定します．

プログラム上のオペランドの記述は，MASMは「DEST，SOU」あるいは「DEST，COUNT」となります．また，gasではオペランドの記述順序が逆になり「SOU，DEST」あるいは「COUNT，DEST」と記述します．

MMXレジスタの8本のレジスタは，物理レジスタ番号に対応する形で，MM0～MM7と名前が付けられています．MASMでは，このMM0～MM7がそのままレジスタ名として

〔リスト1〕MASMのMMX命令の記述例

```
                            .586
                            .mmx

                            .model flat

00000000                    .data

00000000 00000000           m32d     dd      0
00000004                    m64q     dq      0
         0000000000000000

0000000C  00000008 [        m64ub8   db      8 dup(0)
           00
          ]
00000014  00000004 [        m64uw4   dw      4 dup(0)
           0000
          ]
0000001C  00000002 [        m64ud2   dd      2 dup(0)
           00000000
          ]

00000024  00000008 [        m64sb8   sbyte   8 dup(0)
           00
          ]
0000002C  00000004 [        m64sw4   sword   4 dup(0)
           0000
          ]
00000034  00000002 [        m64sd2   sdword  2 dup(0)
           00000000
          ]


00000000                    .code
00000000  0F 6E C0                   movd    mm0,eax
00000003  0F 6E CA                   movd    mm1,edx
00000006  0F 6E 15                   movd    mm2,m32d
          00000000 R
0000000D  0F 7E C0                   movd    eax,mm0
00000010  0F 7E CA                   movd    edx,mm1
00000013  0F 7E 15                   movd    m32d,mm2
          00000000 R

0000001A  0F 7F C8                   movq    mm0,mm1
0000001D  0F 6F 0D                   movq    mm1,qword ptr m64ub8
          0000000C R
00000024  0F 6F 15                   movq    mm2,qword ptr m64sb8
          00000024 R
0000002B  0F 6F 1D                   movq    mm3,qword ptr m64uw4
          00000014 R
00000032  0F 6F 25                   movq    mm4,qword ptr m64sw4
          0000002C R
00000039  0F 6F 2D                   movq    mm5,qword ptr m64ud2
          0000001C R
00000040  0F 6F 35                   movq    mm6,qword ptr m64sd2
          00000034 R
00000047  0F 6F 3D                   movq    mm7,m64q
          00000004 R
0000004E  0F 7F C1                   movq    mm1,mm0
00000051  0F 7F 0D                   movq    qword ptr m64ub8,mm1
          0000000C R
00000058  0F 7F 15                   movq    qword ptr m64sb8,mm2
          00000024 R
0000005F  0F 7F 1D                   movq    qword ptr m64uw4,mm3
          00000014 R
00000066  0F 7F 25                   movq    qword ptr m64sw4,mm4
          0000002C R
0000006D  0F 7F 2D                   movq    qword ptr m64ud2,mm5
          0000001C R
00000074  0F 7F 35                   movq    qword ptr m64sd2,mm6
          00000034 R
0000007B  0F 7F 3D                   movq    m64q,mm7
          00000004 R

                            ～中略～

000002D8  0F EF 1D                   pxor    mm3,qword ptr m64uw4
          00000014 R
000002DF  0F EF 25                   pxor    mm4,qword ptr m64sw4
          0000002C R
000002E6  0F EF 2D                   pxor    mm5,qword ptr m64ud2
          0000001C R
000002ED  0F EF 35                   pxor    mm6,qword ptr m64sd2
          00000034 R
000002F4  0F EF 3D                   pxor    mm7,m64q
          00000004 R
;----------------------------
000002FB  0F F1 C1                   psllw   mm0,mm1
000002FE  0F F1 15                   psllw   mm2,m64q
          00000004 R
00000305  0F 71 F3/ 0                psllw   mm3,5
00000309  0F F2 C1                   pslld   mm0,mm1
0000030C  0F F2 15                   pslld   mm2,m64q
          00000004 R
00000313  0F 72 F3/ 0                pslld   mm3,5
00000317  0F F3 C1                   psllq   mm0,mm1
0000031A  0F F3 15                   psllq   mm2,m64q
          00000004 R
00000321  0F 73 F3/ 0                psllq   mm3,5

00000325  0F D1 C1                   psrlw   mm0,mm1
00000328  0F D1 15                   psrlw   mm2,m64q
          00000004 R
0000032F  0F 71 D3 05                 psrlw   mm3,5
00000333  0F D2 C1                   psrld   mm0,mm1
00000336  0F D2 15                   psrld   mm2,m64q
          00000004 R
0000033D  0F 72 D3 05                 psrld   mm3,5
00000341  0F D3 C1                   psrlq   mm0,mm1
00000344  0F D3 15                   psrlq   mm2,m64q
          00000004 R
0000034B  0F 73 D3 05                 psrlq   mm3,5

0000034F  0F E1 C1                   psraw   mm0,mm1
00000352  0F E1 15                   psraw   mm2,m64q
          00000004 R
00000359  0F 71 E3 05                 psraw   mm3,5
0000035D  0F E2 C1                   psrad   mm0,mm1
00000360  0F E2 15                   psrad   mm2,m64q
          00000004 R
00000367  0F 72 E3 05                 psrad   mm3,5
                            ;----------------------------
0000036B  0F 77                      emms
                            end
```

〔リスト2〕gasのMMX命令の記述例

```
  1                     .data
  2
  3 0000 00000000       m32a:   .long   0
  4 0004 00000000       m32b:   .long   0
  5 0008 00000000       m64a:   .quad   0
  5      00000000
  6 0010 00000000       m64b:   .quad   0
  6      00000000
  7
  8                     .text
  9 0000 0F6EC0                 movd    %eax,%mm0
 10 0003 0F6ECA                 movd    %edx,%mm1
 11 0006 0F6E1500               movd    m32a,%mm2
 11      000000
 12 000d 0F6E1D04               movd    m32b,%mm3
 12      000000
 13 0014 0F7EC0                 movd    %mm0,%eax
 14 0017 0F7EC8                 movd    %mm1,%eax
 15 001a 0F7E1500               movd    %mm2,m32a
 15      000000
 16 0021 0F7E1D04               movd    %mm3,m32b
 16      000000
 17
 18 0028 0F6FC1                 movq    %mm1,%mm0
 19 002b 0F6F1508               movq    m64a,%mm2
 19      000000
 20 0032 0F6F1D10               movq    m64b,%mm3
 20      000000
 21 0039 0F6FC8                 movq    %mm0,%mm1
 22 003c 0F7F1508               movq    %mm2,m64a
 22      000000
 23 0043 0F7F1D10               movq    %mm3,m64b
 23      000000
 24                     #---------------------------
 25 004a 0F63C1                 packsswb   %mm1,%mm0
 26 004d 0F631508               packsswb   m64a,%mm2
 26      000000
 27 0054 0F6BC1                 packssdw   %mm1,%mm0
 28 0057 0F6B1508               packssdw   m64a,%mm2
 28      000000
 29
 30 005e 0F67C1                 packuswb   %mm1,%mm0
 31 0061 0F671508               packuswb   m64a,%mm2
 31      000000
 32
 33 0068 0F68C1                 punpckhbw  %mm1,%mm0
 34 006b 0F681508               punpckhbw  m64a,%mm2
 34      000000
 35 0072 0F69C1                 punpckhwd  %mm1,%mm0
 36 0075 0F691508               punpckhwd  m64a,%mm2
 36      000000
 37 007c 0F6AC1                 punpckhdq  %mm1,%mm0
 38 007f 0F6A1508               punpckhdq  m64a,%mm2
 38      000000
 39
 40 0086 0F60C1                 punpcklbw  %mm1,%mm0
 41 0089 0F601508               punpcklbw  m64a,%mm2
 41      000000
 42 0090 0F61C1                 punpcklwd  %mm1,%mm0
 43 0093 0F611508               punpcklwd  m64a,%mm2
 43      000000
 44 009a 0F62C1                 punpckldq  %mm1,%mm0
 45 009d 0F621508               punpckldq  m64a,%mm2
 45      000000
 46                     #---------------------------
 47 00a4 0FFCC1                 paddb   %mm1,%mm0
 48 00a7 0FFC1508               paddb   m64a,%mm2
 48      000000
 49 00ae 0FFDC1                 paddw   %mm1,%mm0
 50 00b1 0FFD1508               paddw   m64a,%mm2
 50      000000
 51 00b8 0FFEC1                 paddd   %mm1,%mm0
 52 00bb 0FFE1508               paddd   m64a,%mm2
 52      000000
 53
 54 00c2 0FECC1                 paddsb  %mm1,%mm0
 55 00c5 0FEC1508               paddsb  m64a,%mm2
 55      000000
 56 00cc 0FEDC1                 paddsw  %mm1,%mm0
 57 00cf 0FED1508               paddsw  m64a,%mm2
 57      000000
 58
 59 00d6 0FDCC1                 paddusb  %mm1,%mm0
 60 00d9 0FDC1508               paddusb  m64a,%mm2
 60      000000
 61 00e0 0FDDC1                 paddusw  %mm1,%mm0
 62 00e3 0FDD1508               paddusw  m64a,%mm2
 62      000000
 63
 64 00ea 0FF8C1                 psubb   %mm1,%mm0
 65 00ed 0FF81508               psubb   m64a,%mm2
 65      000000
 66 00f4 0FF9C1                 psubw   %mm1,%mm0
 67 00f7 0FF91508               psubw   m64a,%mm2
 67      000000
 68 00fe 0FFAC1                 psubd   %mm1,%mm0
 69 0101 0FFA1508               psubd   m64a,%mm2

                          ～中略～

 91      000000
 92 0162 0F76C1                 pcmpeqd  %mm1,%mm0
 93 0165 0F761508               pcmpeqd  m64a,%mm2
 93      000000
 94
 95 016c 0F64C1                 pcmpgtb  %mm1,%mm0
 96 016f 0F641508               pcmpgtb  m64a,%mm2
 96      000000
 97 0176 0F65C1                 pcmpgtw  %mm1,%mm0
 98 0179 0F651508               pcmpgtw  m64a,%mm2
 98      000000
 99 0180 0F66C1                 pcmpgtd  %mm1,%mm0
100 0183 0F661508               pcmpgtd  m64a,%mm2
100      000000
101                     #---------------------------
102 018a 0FDBC1                 pand    %mm1,%mm0
103 018d 0FDB1508               pand    m64a,%mm2
103      000000
104 0194 0FDFC1                 pandn   %mm1,%mm0
105 0197 0FDF1508               pandn   m64a,%mm2
105      000000
106 019e 0FEBC1                 por     %mm1,%mm0
107 01a1 0FEB1508               por     m64a,%mm2
107      000000
108 01a8 0FEFC1                 pxor    %mm1,%mm0
109 01ab 0FEF1508               pxor    m64a,%mm2
109      000000
110                     #---------------------------
111 01b2 0FF1C1                 psllw   %mm1,%mm0
112 01b5 0FF11508               psllw   m64a,%mm2
112      000000
113 01bc 0F71F305               psllw   $5,%mm3
114 01c0 0FF2C1                 pslld   %mm1,%mm0
115 01c3 0FF21508               pslld   m64a,%mm2
115      000000
116 01ca 0F72F305               pslld   $5,%mm3
117 01ce 0FF3C1                 psllq   %mm1,%mm0
118 01d1 0FF31508               psllq   m64a,%mm2
118      000000
119 01d8 0F73F305               psllq   $5,%mm3
120
121 01dc 0FD1C1                 psrlw   %mm1,%mm0
122 01df 0FD11508               psrlw   m64a,%mm2
122      000000
123 01e6 0F71D305               psrlw   $5,%mm3
124 01ea 0FD2C1                 psrld   %mm1,%mm0
125 01ed 0FD21508               psrld   m64a,%mm2
125      000000
126 01f4 0F72D305               psrld   $5,%mm3
127 01f8 0FD3C1                 psrlq   %mm1,%mm0
128 01fb 0FD31508               psrlq   m64a,%mm2
128      000000
129 0202 0F73D305               psrlq   $5,%mm3
130
131 0206 0FE1C1                 psraw   %mm1,%mm0
132 0209 0FE11508               psraw   m64a,%mm2
132      000000
133 0210 0F71E305               psraw   $5,%mm3
134 0214 0FE2C1                 psrad   %mm1,%mm0
135 0217 0FE21508               psrad   m64a,%mm2
135      000000
136 021e 0F72E305               psrad   $5,%mm3
137                     #---------------------------
138 0222 0F77                   emms
```

ニモニックはCPUのマニュアルで使われている表記をそのまま使用する

転送元(SOU)

転送先(DEST)

シフトのカウント(COUNT)指定

使われています．gasでは%を前に付けた%mm0，%mm1，…，%mm7がMMXレジスタの指定となります．

実際のMASMでのMMX命令の記述例を**リスト1**(p.158)，gasを**リスト2**(p.159)に示します．

● ラップアラウンド処理と飽和処理

MMX命令は，変換，演算を含む，すべての命令が実行に際してEFLAGS上のフラグを変化させません．そのため，変換や演算でオーバフローが発生しても，それをフラグから得ることはできません．

MMX命令使用時のオーバフローがどうしても心配な場合は，事前に演算に使用する値をチェックするか，オーバフローが発生しないビット数の多い型で演算するといった配慮が必要となります．しかし，MMX命令は実行してもEFLAGS上のフラグを変化させない代わりに，オーバフローの対策として命令によりラップアラウンド処理と飽和処理が選択できるようになっています．

(1) ラップアラウンド処理

*n*ビット同士の加減算で，演算結果が*n*ビットに納まらずオーバフローした場合，オーバフローした上位のビットは無視し，下位の*n*ビットのみを有効な値とするのがラップアラウンド処理です．

たとえば，ワード整数同士の加算78FDh＋952Chは，10E29hという結果になりますが，ラップアラウンドで加算すると0E29hというワード整数の値が加算結果として得られることになります．

(2) 飽和(Saturation)処理

飽和処理では，これから転送先に転送しようとしている値(転送元の値や演算結果)が，転送先の一つの要素で表せる範囲の値を超えていた場合，転送先に転送しようとしている値を，転送先で扱える範囲の最大値あるいは最小値で置き換えるのが飽和処理です．

飽和処理は，転送先の要素がバイト整数あるいはワード整数のとき使用できます．具体的な処理は次のようになります．

① 転送先の要素が符号なしバイト整数の場合

これから転送先に転送しようとしている値が255(FFh)より大きければ，その値を255(FFh)で置き換えます．

また，これから転送先に転送しようとしている値が負であれば，その値を0(00h)で置き換えます．

② 転送先の要素が符号付きバイト整数の場合

これから転送先に転送しようとしている値が+127(7Fh)より大きければ，その値を+127(7Fh)で置き換えます．

また，これから転送先に転送しようとしている値が－128(80h)より小さければ，その値を－128(80h)で置き換えます．

③ 転送先の要素が符号なしワード整数の場合

これから転送先に転送しようとしている値が65535(FFFFh)より大きければ，その値を65535(FFFFh)で置き換えます．

また，これから転送先に転送しようとしている値が負であれば，その値を0(0000h)で置き換えます．

④ 転送先の要素が符号付きワード整数の場合

これから転送先に転送しようとしている値が+32767(7FFFh)より大きければ，その値を+32767(7FFFh)で置き換えます．

また，これから転送先に転送しようとしている値が－32768(8000h)より小さければ，その値を－32768(8000h)で置き換えます．

MMXレジスタに定数をセットする方法

MMX命令には，MMXレジスタに定数を設定するイミディエイト命令がありません．そのため，定数はすべてメモリ上に置く必要があります．

ただし，MMXの既存の命令を組み合わせることで，MMXレジスタのゼロクリアや，MMXレジスタの全ビットを1にする，MMXレジスタ上の各要素をすべて値(1)に設定することができます．

(1) MMXレジスタのゼロクリア

PXOR命令を使うことで，任意のMMXレジスタをゼロクリアすることができます．たとえば，

```
PXOR   MM2, MM2
```

とすることでMM2レジスタがゼロ(全ビットが0)の状態になります．

(2) MMXレジスタの全ビットを1する

PCMPEQB命令を使用することで，任意のMMXレジスタを全ビット1にすることができます．たとえば，

```
PCMPEQB   MM4, MM4
```

とすることでMM4レジスタの全ビットが1の状態になります．

また，この全ビットを1の状態にするということは，各要素を符号付き整数の値(－1)に設定することにもなります．

(3) MMXレジスタ上の各要素をすべて値(1)に設定する

今述べた，PXOR命令とPCMPEQB命令，そしてPSUB?命令を使用することで，任意のMMXレジスタ上の各要素をすべて値の1に設定できます．

たとえば，

```
PXOR      MM0, MM0
PCMPEQB   MM1, MM1
PSUBB     MM0, MM1
```

とすることで，MM0上のパックドバイト整数の各要素をすべて値(1)にすることができます．

ちなみに，最後のPSUBB命令をPSUBW命令にすることでパックドワード整数の各要素をすべて値(1)に，PSUBD命令にすることで，パックドダブルワード整数の各要素をすべて値(1)にすることができます．

〔**図7**〕**転送命令の動作**

MOVD　mm，r/m32

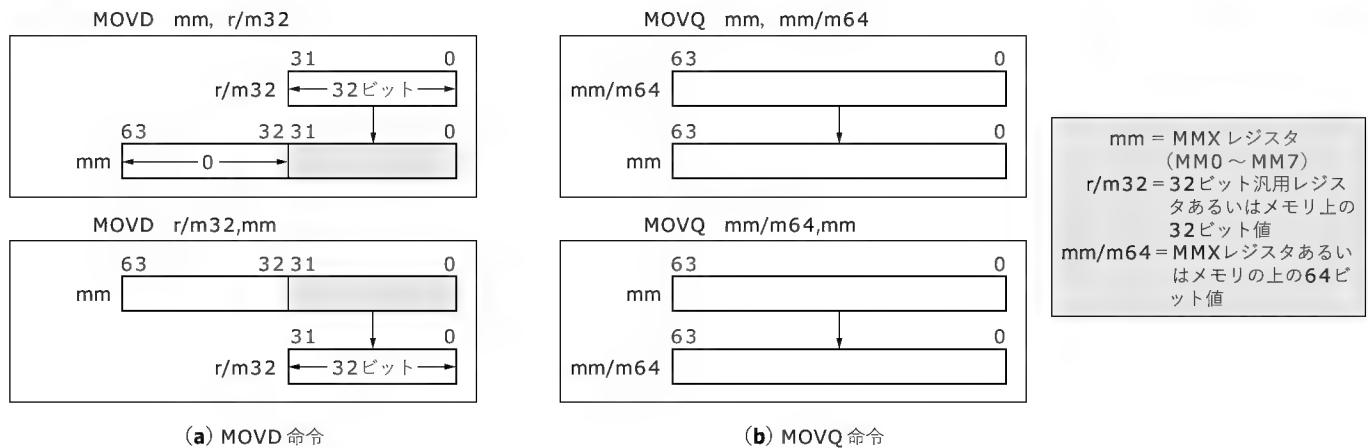
MOVD　r/m32,mm

(**a**) MOVD命令

MOVQ　mm，mm/m64

MOVQ　mm/m64,mm

(**b**) MOVQ命令

mm＝MMXレジスタ（MM0～MM7）
r/m32＝32ビット汎用レジスタあるいはメモリ上の32ビット値
mm/m64＝MMXレジスタあるいはメモリの上の64ビット値

MMX命令の各命令の動作

MMX命令は，**表1**に示すように機能別に「データ転送命令」，「変換命令」，「パックド算術命令」，「比較命令」，「論理演算命令」，「シフト命令」，「ステート管理命令」の7種類に分類されています．

この分類にしたがい各MMX命令の動作を説明します．

● データ転送命令

データ転送命令は，ダブルワード，クワッドワードの値を転送するための命令で，MOVDとMOVQの二つの命令があります．

(1) MOVD命令

ダブルワードの値を転送元から転送先にコピーします．

転送元としてはCPUの32ビット汎用レジスタ，あるいはメモリ上のダブルワードの値を指定した場合，転送先にはMMXレジスタ(MM0～MM7)を指定します．このとき転送元のダブルワードの値は，32ビットの符号なしと考え，ゼロ拡張した64ビット長の符号なしクワッドワード値がMMXレジスタに格納されます．

逆に，転送元にMMXレジスタ(MM0～MM7)を指定した場合は，転送先にはCPUの32ビット汎用レジスタ，あるいはメモリ上のダブルワードの領域を指定します．この場合，指定MMXレジスタの下位32ビットの値のみが，そのままCPUの32ビット汎用レジスタ，あるいはメモリ上のダブルワードの領域にストアされます．

図7(**a**)は，MOVD命令の動作を図で表したものです．

(2) MOVQ命令

64ビット(クワッドワード)の値を転送元から転送先にコピーします．「MMXレジスタ←MMXレジスタ」，「MMXレジスタ←メモリ上のクワッドワード」，「メモリ上のクワッドワード←MMXレジスタ」の3種類の転送が使用できます〔**図7**(**b**)〕．

● 変換命令

変換命令は，パックさた値を別の形式のパックされた値に変換するための命令です．命令にはPACKSSWB/PACKSSDW/PACKUSWB，PUNPCKHBW/PUNPCKHWD/PUNPCKHDQ，PUNPCKLBW/PUNPCKLWD/PUNPCKLDQの9命令があります．

(1) PACKSSWB/PACKSSDW/PACKUSWB命令

PACKSSWB命令は，転送先と転送元のパックド符号付きワード整数から，8ワードをリードし，飽和処理して得られた符号付きバイト値8個を，パックド符号付きバイト整数として転送先にストアします〔**図8**(**a**)〕．

PACKSSDW命令は，転送先と転送元のパックド符号付きダブルワード整数から，4ダブルワードをリードし，飽和処理して得られた符号付きワード値4個を，パックド符号付きワード整数として転送先にストアします〔**図8**(**b**)〕．

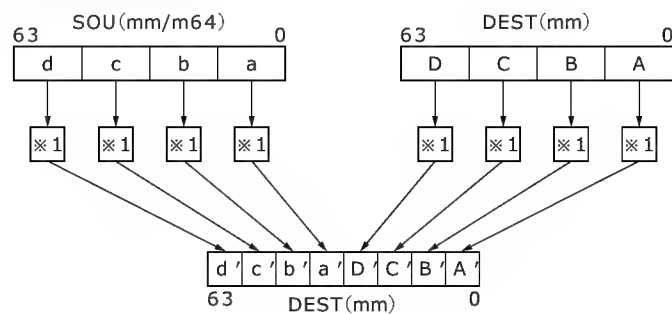
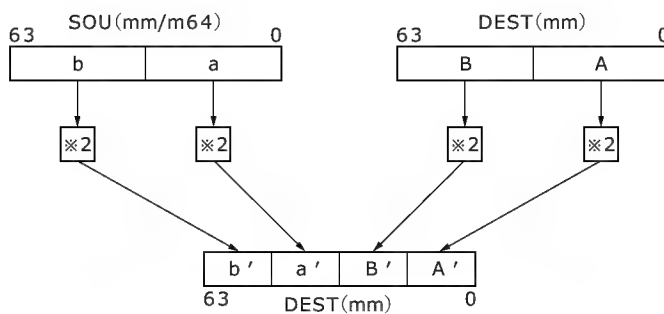
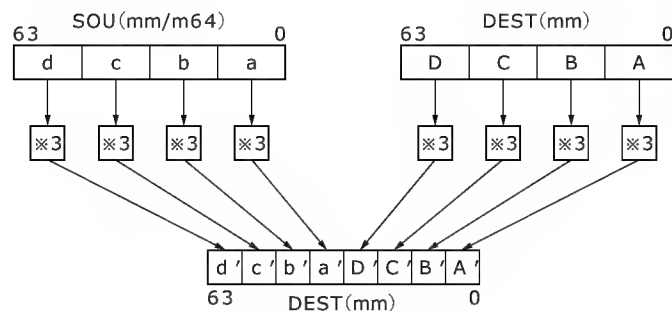
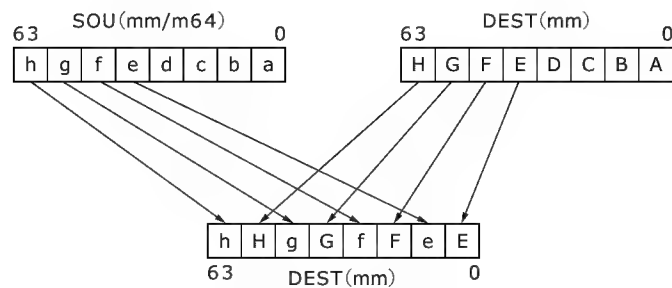
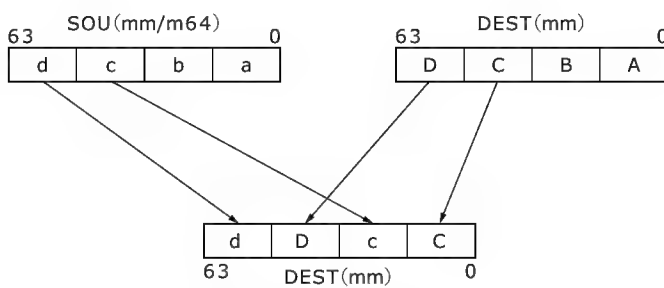
PACKUSWB命令は，転送先と転送元のパックド符号付きワード整数から，8ワードをリードし，飽和処理して得られた符号なしバイト値8個を，パックド符号なしバイト整数として転送先にストアします〔**図8**(**c**)〕．

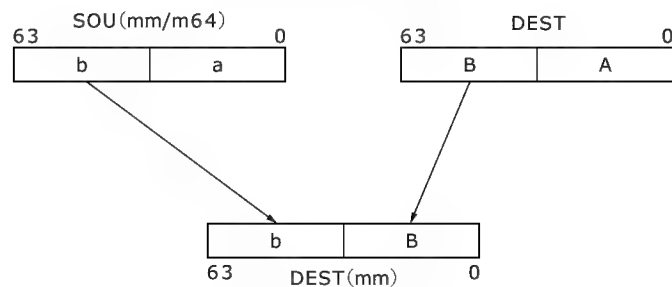
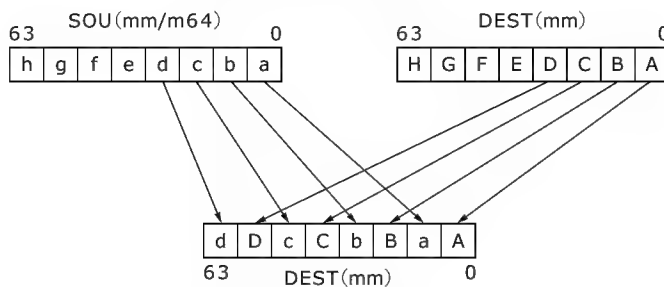
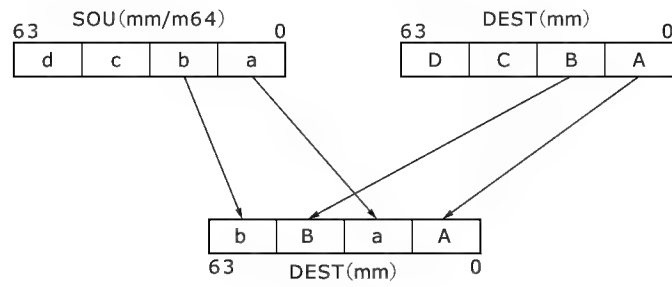
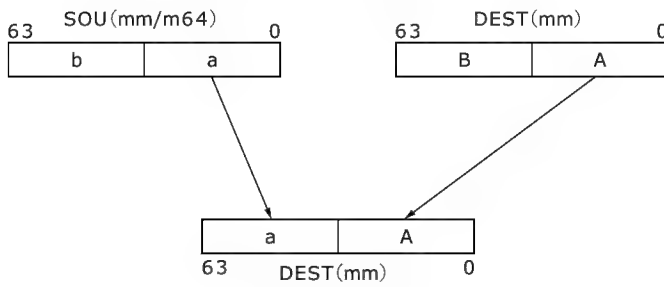
(2) PUNPCKHBW/PUNPCKHWD/PUNPCKHDQ命令

PUNPCKHBW命令は，転送先の上位32ビットと転送元の上位32ビットのパックドバイト整数から，8バイト抽出し，インタリーブして転送先にストアします．実際の動作は**図8**(**d**)のようになります．

PUNPCKHWD命令は，転送先の上位32ビットと転送元の上位32ビットのパックドワード整数から，4ワード抽出し，インタリーブして転送先にストアします．実際の動作は**図8**(**e**)のようになります．

PUNPCKHDQ命令は，転送先の上位32ビットと転送元の上位32ビットのパックドダブルワード整数から，2ダブルワード抽出し，インタリーブして転送先にストアします．実際の動作は**図8**(**f**)のようになります．

〔図8〕変換命令の動作

(a) PACKSSWB命令

(b) PACKSSDW命令

(c) PACKUSWB命令

(d) PUNPCKHBW命令

(e) PUNPCKHWD命令

(f) PUNPCKHDQ命令

(g) PUNPCKLBW命令

(h) PUNPCKLWD命令

(i) PUNPCKLDQ命令

(3) PUNPCKLBW/PUNPCKLWD/PUNPCKLDQ 命令

PUNPCKLBW命令は，転送先の下位32ビットと転送元の下位32ビットのパックドバイト整数から，8バイト抽出し，インタリーブして転送先にストアします．実際の動作は**図8(g)**のようになります．

PUNPCKLWD命令は，転送先の下位32ビットと転送元の下位32ビットのパックドワード整数から，4ワード抽出し，インタリーブして転送先にストアします．実際の動作は**図8(h)**のようになります．

PUNPCKLDQ命令は，転送先の下位32ビットと転送元の下位32ビットのパックドダブルワード整数から，2ダブルワード抽出し，インタリーブして転送先にストアします．実際の動作は**図8(i)**のようになります．

● パックド算術命令

パックド算術命令は，パックされた値の加減乗算を行うものです．ただしMMXには，残念ながらパックされた値の除算は用意されていません．加算としてPADDB/PADDW/PADDD，PADDSB/PADDSW/PADDUSB/PADDUSWの7命令，減算としてPSUBB/PSUBW/PSUBD，PSUBSB/PSUBSW/PSUBUSB/PSUBUSWの7命令，乗算としてPMULHW/PMULLW，PMADDWDの3命令があります．

(1) PADDB/PADDW/PADDD 命令

転送先のパックされた値に，転送元のパックされた値を，各要素ごとに加算し，その和をラップアラウンド処理し，転送先にストアします．PADDB命令がパックドバイト整数，PADDW命令がパックドワード整数，PADDD命令がパックドダブルワード整数のラップアラウンド加算です．

これらのラップアラウンド加算の命令は，符号付き符号なしの両方の整数で使用できますが，先にも述べたようにMMX命令は実行してもフラグをセットしないので，オーバフローや桁上がりのキャリの発生が，命令実行後ではわからないので，使用には注意が必要です．

(2) PADDSB/PADDSW/PADDUSB/PADDUSW 命令

転送先のパックされた値に，転送元のパックされた値を，各要素ごとに加算し，その和を飽和処理し，転送先にストアします．PADDSB命令がパックド符号付きバイト整数，PADDSW命令がパックド符号付きワード整数，PADDUSB命令がパックド符号なしバイト整数，PADDUSW命令がパックド符号なしワード整数の飽和加算です．

(3) PSUBB/PSUBW/PSUBD 命令

転送先のパックされた値から，転送元のパックされた値を，各要素ごとに減算し，その差をラップアラウンド処理し，転送先にストアします．PSUBB命令がパックドバイト整数，PSUBW命令がパックドワード整数，PSUBD命令がパックドダブルワード整数のラップアラウンド減算です．

これらのラップアラウンド減算の命令は，符号付き符号なしの両方の整数で使用できますが，加算のときと同じようにMMX命令は実行してもフラグをセットしないので，オーバフローや桁借りのキャリの発生が，命令実行後ではわからないので，その点加算と同じような注意が必要です．

(4) PSUBSB/PSUBSW/PSUBUSB/PSUBUSW 命令

転送先のパックされた値から，転送元のパックされた値を，各要素ごとに減算し，その差を飽和処理し，転送先にストアします．PSUBSB命令がパックド符号付きバイト整数，PSUBSW命令がパックド符号付きワード整数，PSUBUSB命令がパックド符号なしバイト整数，PSUBUSW命令がパックド符号なしワード整数の飽和減算です．

以上をまとめてパックド算術命令の動作を**図9(a)**～**図9(c)**に示します．

(5) PMULHW/PMULLW 命令

転送先のパックされた値と，転送元のパックされた値を，各要素ごとに符号付きで乗算し，その積(32ビット長)の上位16ビットあるいは下位16ビットを抽出し，転送先にストアします〔**図9(d)**〕．PMULHW命令では，各要素の積の上位16ビットが転送先にストアされます．PMULLW命令では，各要素の積の下位16ビットが転送先にストアされます．

(6) PMADDWD 命令

転送先のパックド符号付きワード整数を[D C B A]，転送元のパックド符号付きワード整数を[d c b a]とすると，PMADDWD命令は，

$$[(D \times d)+(C \times c)\quad(B \times b)+(A \times a)]$$

で表されるパックド符号付きダブルワード整数を計算します〔**図9(e)**〕．ただし，最終的に一つの値となるグループ内の4ワードがすべて，最小値である8000hの場合のみ，その計算結果は8000hとなります．

● 比較命令

比較命令は，パックされた値の各要素ごとに比較するもので，「等しい」と「より大きい」の2種類の比較があります．比較の結果，条件が成立していれば，転送先のその要素のビットをすべて1し，条件が成立していなければその要素のビットをすべて0にします．PCMPEQB，PCMPEQW，PCMPEQD命令が「等しい」の比較，PCMPGTB，PCMPGTW，PCMPGTD命令が「より大きい」の比較です．

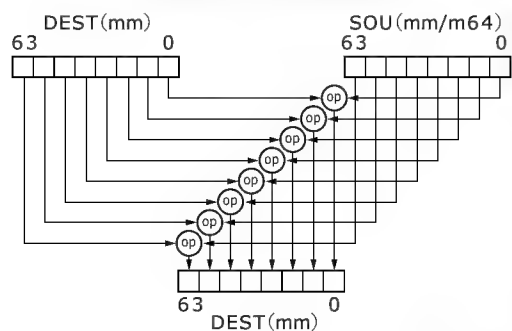
(1) PCMPEQB，PCMPEQW，PCMPEQD 命令

PCMPEQB命令がパックドバイト整数，PCMPEQWがパックドワード整数，PCMPEQD命令がパックドダブルワード整数の比較です．この「等しい」の比較は，その性格上，符号付き符号なしの両方の整数で使用できます．

(2) PCMPGTB，PCMPGTW，PCMPGTD 命令

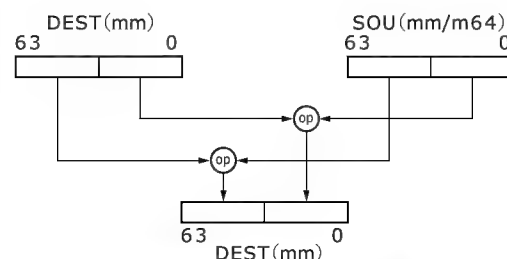
PCMPGTB命令がパックド符号付きバイト整数，PCMPGTWがパックド符号付きワード整数，PCMPGTD命令がパックド符号付きダブルワード整数の比較です．この「より大きい」の比較は，符号付き整数として比較され，要素が(転送先の値)>(転送元の値)のとき条件成立となります．

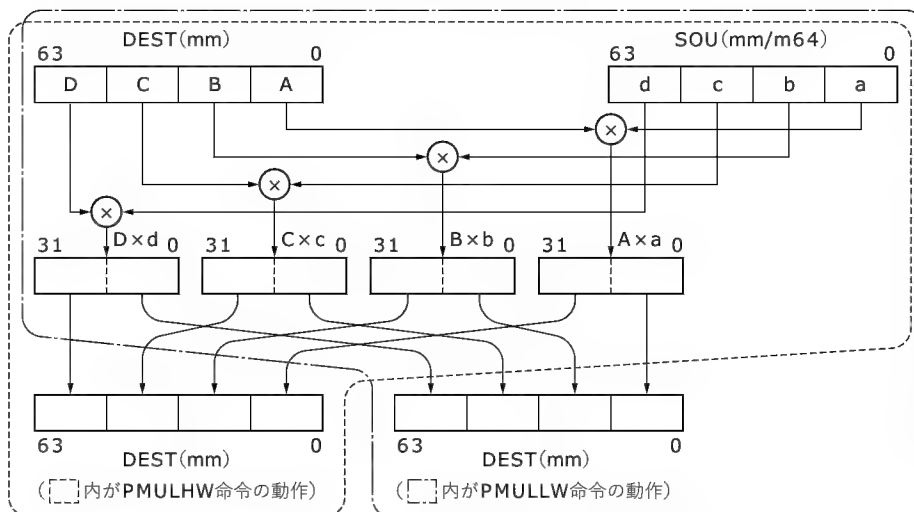
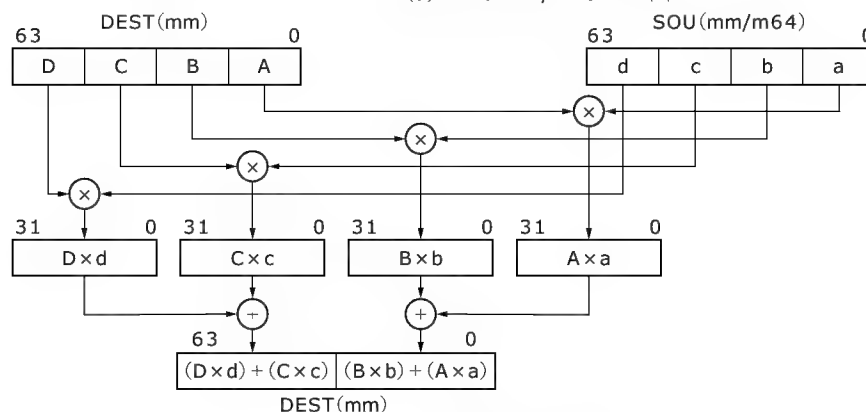
〔図9〕パックド算術命令と比較命令の動作

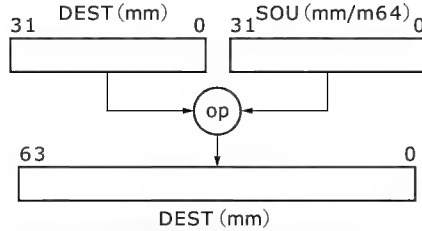
op(演算) DEST ←DEST op SOU	命令
ラップアラウンド加算	PADDB
符号付き飽和加算	PADDSB
符号なし飽和加算	PADDUSB
ラップアラウンド減算	PSUBB
符号付き飽和減算	PSUBSB
符号なし飽和減算	PSUBUSB
比較(=)	PCMPEQB
比較(>)	PCMPGTB

(a) パックドバイト整数の演算

op(演算) DEST ←DEST op SOU	命令
ラップアラウンド加算	PADDW
符号付き飽和加算	PADDSW
符号なし飽和加算	PADDUSW
ラップアラウンド減算	PSUBW
符号付き飽和減算	PSUBSW
符号なし飽和減算	PSUBUSW
比較(=)	PCMPEQW
比較(>)	PCMPGTW

(b) パックドワード整数の演算

DEST(mm) 63 0　SOU(mm/m64) 63 0　op　63 0 DEST(mm)

op(演算) DEST ←DEST op SOU	命令
ラップアラウンド加算	PADDD
ラップアラウンド減算	PSUBD
比較(=)	PCMPEQD
比較(>)	PCMPGTD

(c) パックドダブルワード整数の演算

(d) PMULHW/PMULLW命令

(e) PMADDWD命令

〔図10〕
論理演算の動作

op (演算)	命令
論理積 DEST←DEST and SOU	PAND
否定論理積 DEST←(not DEST) and SOU	PANDN
論理和 DEST←DEST or SOU	POR
排他的論理和 DEST←DEST xor SOU	PXOR

● 論理演算命令

転送先のクワッドワード値に，転送元のクワッドワード値を，ビットごとに論理演算し，その結果を転送先にストアします(**図10**)．演算は，PAND命令の論理積，PANDN命令の否定論理積，POR命令の論理和，PXOR命令の排他的論理和の4種類です．PAND，POR，PXOR命令は，汎用命令のAND，OR，XOR命令と同じ動作をするMMX命令です．

PANDN命令の否定論理積は，転送先をDEST，転送元をSOUで表すと

DEST ← (not DEST) and SOU

の論理演算を行う命令です．この否定論理積は，汎用命令にはなく，SIMD命令でのみ使用できる命令です．

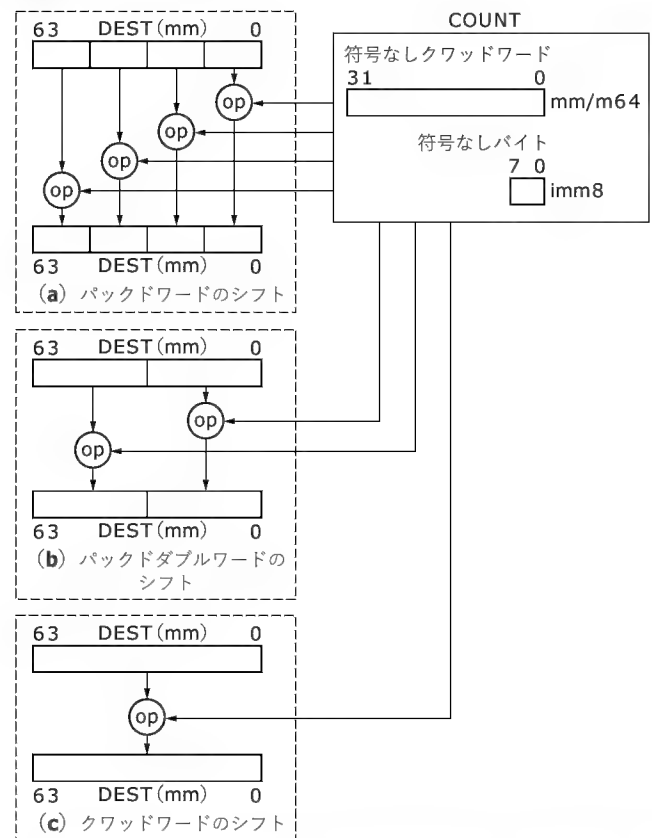
● シフト命令

転送先のパックされた値の各要素の値を，カウントオペランドで指定されたビット数分シフトします(**図11**)．カウントオペランドの有効な値は，転送先がパックドワード整数なら0～15，パックドダブルワード整数なら0～31，クワッドワード整数なら0～63となります．これ以外の値をカウントとして指定すると，ゼロが転送先に設定されます．

PSLLW/PSLLD/PSLLQ命令が論理左シフト，PSRLW/PSRLD/PSRLQ命令が論理右シフト，そしてPSRAW/PSRAD命令が算術右シフトとなります．シフトの動作自体は，汎用命令の論理左シフト，論理右シフト，算術右シフトと同じです．PSLLW，PSRLW，PSRAW命令がパックドワード整数，PSLLD，PSRLD，PSRAD命令がパックドダブルワード整数，PSLLQ，PSRLQ命令がクワッドワード整数のシフトとなります．

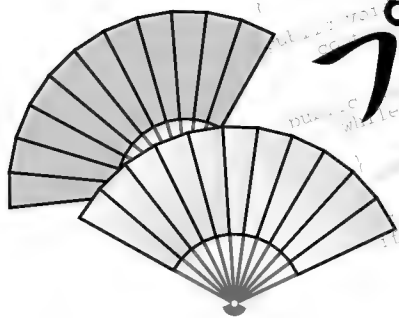
● ステート管理命令

MMXではステート管理命令は，EMMS命令のみ定義されています．一連のMMX命令の実行を終え，FPU命令の実行前には，必ずこのEMMS命令を実行するようにします．

〔図11〕シフト命令の動作

op (演算) DEST←DEST op COUNT	型 (ビット長n)	命令
論理左シフト MSB LSB n−1 n−2 … 1 0 ←0 1ビットづつCOUNT回左へシフトする	ワード (16ビット)	PSLLW
	ダブルワード (32ビット)	PSLLD
	クワッドワード (64ビット)	PSLLQ
論理右シフト MSB LSB n−1 n−2 … 1 0 0→ 1ビットずつCOUNT回右へシフトする	ワード (16ビット)	PSRLW
	ダブルワード (32ビット)	PSRLD
	クワッドワード (64ビット)	PSRLQ
算術右シフト MSB LSB n−1 n−2 … 1 0 S 1ビットずつCOUNT回右へシフトする	ワード (16ビット)	PSRAW
	ダブルワード (32ビット)	PSRAD

* *

次回はSIMD命令の続きとして，SSE，SSE2命令の概要について説明する予定です．

おおぬき・ひろゆき　大貫ソフトウェア設計事務所

宮坂電人

第8回 さまざまなアンチパターンの概要

はじめに

前回に引き続き，参考文献1)から紹介しましょう．第2部の本論では，実際のアンチパターンの症状や対策が詳しく述べられています．アンチパターンが出現する局面で以下の三つにおおまかに分類され，それぞれに1章ずつを割り当てて説明されています．

(A) ソフトウェア開発のアンチパターン
(B) ソフトウェアのアーキテクチャのアンチパターン
(C) プロジェクト管理のアンチパターン

この三つの分類は，前回紹介した「三つの視点からのアンチパターン」に対応しています．それぞれ，

(A) 開発の次元のアンチパターン――問題の中心が技術レベルで，おもにプログラマから生まれる
(B) アーキテクチャの次元のアンチパターン――システムの構造や構成に関わる諸問題
(C) 管理の次元のアンチパターン――ソフトウェア開発/導入の工程管理や開発組織に関連する諸問題

に対応しています．パターンはいくつかあり，それぞれにていねいな解説をすると膨大な分量になるので，項目と症状，ソフトウェアの再構想による解決策(再構想解)を簡潔に紹介するにとどめます．より詳細なことは『アンチパターン―ソフトウェア危篤患者の救出』[1]を参照してください．

ソフトウェア開発のアンチパターン(第5章)

ここで述べられているアンチパターンは文字どおり，ソフトウェア開発の現場で直視されるアンチパターンで，おそらく本誌の読者にもっとも身近でありもっとも深刻に状況を理解できるものでしょう．他の章で述べられているものと比較してもっとも手をつけやすいのですが，もっとも身近であるがゆえ，ついつい状況に流され案外対策しにくいものかもしれません．

● 肥満児

1個の巨大クラス(肥満児クラス)に処理が集中している状況です．他のクラスは貧弱な構造体と化していて肥満児クラスに操られているだけの存在で，よく観察するとオブジェクト指向ではなく手続き指向の延長線上でプログラムが設計，実装されている状態です．肥満児アンチパターンはオブジェクト指向プログラミングが苦手なプログラマや初心者によく見受けるパターンです．

▶**再構想解**：肥満児クラスに集中している機能を他のクラスに移行できないかを検討してみます．あるいは移行の受け皿となるクラスを複数用意すべきかもしれません．手続き指向的な設計，実装によって肥満児クラスはちょうどメインルーチンであるかのようにふるまっているため，機能の分割という視点はサブルーチンの分割に勘違いしてしまう可能性があり，後で述べる「機能的分解アンチパターン」にはまりこむ危険性があります．関連するメソッドや関連するデータ群に沿って，機能群ではなくオブジェクトやクラスの再構成という方向で再検討をする必要があります．

● 絶え間なき陳腐化

急速な技術変化に追い付こうとして追い付けていない状況です．自分のところの製品やプログラムのバージョンアップはもちろんのこと，他社のバージョンアップによっても対応が難しくなっている状態です．

▶**再構想解**：安定的な技術と制御可能なインターフェースのみを使います．一私企業が決めた規格ではなく複数組織が合意の上で決めたオープンシステムの規格を採用することで実現しやすくなるでしょう．

● 溶岩の流れ

使われているのかむだなのかわからないコードやモジュールが作業中のコードに存在しており，手をつけていいのか悪いのかわからないので放置されている状況です．しかし放置することは問題の本質的な解決ではなく，先延ばしにしているだけの状態です．

▶**再構想解**：予防策としては，アーキテクチャベースに開発過程を制御することです．一生懸命にコーディングしていくというスタイルではなく．また，むだなコードの除去によってバグが導入されることがありますが，あわてて対症療法的に修理したくなる誘惑に耐え，コードの各部分の依存関係を調べアーキテクチャレベルで現象をとらえ対策することです．

● 曖昧な視点

分析の視点と実装の視点が混乱したまま適用されることで分

析も設計も使い物にならなくなる状況です．たとえばオブジェクト指向の場合，インターフェースと実装の分離という重要な原則が破綻し，せっかくのオブジェクト指向のメリットが生かせなくなります．

▶**再構想解：**オブジェクト指向分析とオブジェクト指向設計においては，三つの視点のいずれであるかを明確にしておくべきです．それは「ビジネス」，「仕様」，「実装」の三つです．

ビジネスの視点とは，ユーザー側の情報と処理をとらえたものです．

仕様の視点とは，ソフトウェアのインターフェースに焦点を当てたものです．この視点ではソフトウェアをどのように実装しているかはわかりません．オブジェクトの外側（利用者側）から見た視点です．

実装の視点はオブジェクトの内側の視点であり，オブジェクトの詳細な内部実装を考えます．

● 機能的分解

オブジェクト指向になっておらず，文字どおり機能的な分解や分割になっている状況です．クラスは単にモジュールや名前空間にすぎず，メソッドは手続き指向のルーチンや関数と何ら変わりません．オブジェクト指向を理解していないか不慣れであるものの手続き指向ではベテランの開発者が陥りやすいアンチパターンです．

▶**再構想解：**これに対しては即効性のある解はなく，開発者自身が「まともな」オブジェクト指向を理解して設計と実装ができるように期待するしかありません．そして，問題のあるプログラムを再構成/再設計する必要があります．

● お邪魔妖怪

コードの中に一時的に現れるクラスがあり，それがたいした役割を果たすわけではなく（たいていはデータを受け渡しするだけとか，何かのラッパらしき挙動をするだけ）むしろ資源のむだ使いや実行効率を低下させるだけの状況です．

▶**再構想解：**お邪魔妖怪クラスが担っていた機能をもっと長寿命のクラスにまかせられないか検討します．開発者がしっかりした分析/設計を行わず自分の勘と経験で（というよりも「行き当たりばったり」で）作業をしたときに出やすい傾向があるので，まず腰をすえてしっかりした分析/設計を行うべきでしょう．

● 蛇足

開発プロジェクトがまったく使われることのない製品を購入し，それが役に立たなかったり，それどころか足を引っぱってしまう状況です．

▶**再構想解：**実際に購入する前に，むだであるかどうかを有能な人材に評価してもらうことです．とりあえず買っておけば何かの保険代わりになるだろうとあやふやな期待などしないことです．

● 打ち出の小槌

開発者が自分の得意な手法にこだわり，明らかに不適切であるにもかかわらず失敗をしたり不経済な行動に陥る状況です．自分のレパートリが少ないから，というだけでなく特定の製品や技術を偏愛するあまり「マニア」となってしまうことによっても起きる事態です．

▶**再構想解：**開発者が新技術を勉強することです．そのためには管理職の協力や支援体制も必要でしょう．開発者の専門知識の向上は，開発者自身のみならず組織にとっても「投資」となりうることも知るべきです．

● 梯子外し

商用製品をカスタマイズしたり再利用コンポーネントを部分変更したとき，元の開発者のサポートが期待できない（会社の倒産や開発者の行方不明などで）ため，どうにも身動きがとれなくなる状況です．

▶**再構想解：**商用製品のカスタマイズや再利用コンポーネントの変更を避けます．あるいは対顧客態度が良心的で絶対倒産しそうもない有力企業の商用規格か，業界の主流的な標準技術を採用することで梯子外しを予防します[注1]．カスタマイズや変更が避けられない場合は，「隔離層」などを使って開発ソフトと商用製品/再利用コンポーネントを分離するのも手です．

● スパゲティコード

文字どおりコードがスパゲティ状に混乱している状況です．場当たり的な対応によって拡張や変更がなされているため，いずれ身動きが取れなくなり自滅に追い込まれるでしょう．

▶**再構想解：**コードのクリーニングによりソフトウェアを再構築します．理想的には，コードのクリーニングは開発工程に最初から標準的部分として盛り込むべきでしょう．このアンチパターンを避ける最良の方法は，スパゲティにならないよう予防すること，つまり書く前に考えて行動計画を立てることです．不幸にも，すでに存在するスパゲティをいじる場合は，それまでの方法を繰り返すのではなく再構築に時間をかけることです．

● 入力クラッジ

プログラムでの入力の取り扱いが粗雑なため，不都合な入力によって簡単に挙動が怪しくなったりクラッシュする状況です．

▶**再構想解：**「製品品質」の入力アルゴリズムの利用を検討します．

● 地雷原

ユーザーレベルでの充分なテストや検証がなされないまま見切り発車されるため，脆弱でバグだらけのソフトウェアが出荷される状況です．

▶**再構想解：**ソフトウェアの試験への適切な投資が必要です．一部の先進的な企業では，プログラミング部門よりも試験部門の人員が多いほどです．開発者はついついプログラムの生産に興味がいき，それがすべてであるかのように思い込みますが，きちんとした試験や「テストケース」，「テストスーツ（試験用プ

注1：絶対倒産しそうもない有力企業であっても「絶え間なき陳腐化」，「砂上の楼閣」で梯子外しをされる場合があるので要注意．

ログラム集)」の充実も大事です．

● 切り貼りプログラミング

ソースコードを一から作成するのが面倒とばかりに，既存のコードで使えそうな部分をまるまるコピーしてもち込む安直な「再利用」がされている状況です．もちろん，これは正当な意味での再利用などではなく単なる手抜きにすぎません．ところがこうした行為が非難されるのではなく，簡単にプログラムを作成するためのノウハウと勘違いされてチーム全員に悪影響を広めてしまったり，コピーした元の部分にバグがあった場合，バグを伝播させてしまうことがあります．

▶**再構想解：**切り貼りプログラミングは，再利用が「ホワイトボックス的」なものである環境で起こりがちです．すべてが白日のもとにさらされ，わからないことがあれば作った人に聞くよりも直接ソースコードを読めばいいという風潮です．これは一見良いことのように思えますが，仕様と実装の混同という悪癖に陥る傾向があります．また仕様をドキュメントに書くのが面倒で，仕様はソースコードを読めという風潮で，業務がまともに遂行できない危険性があります．とくに対外的な面で支障をきたしたり「あそこの会社はドキュメントの書き方すら知らない」とあざけられることがあります．

ブラックボックス的な再利用，つまりどのように実装しているかをソースコードから読むのではなく，どのようなインターフェースであるかをドキュメントに記述し，インターフェースの仕様を手がかりにした再利用や拡張を行う体制を作るべきです．

● 暗室栽培

開発者とエンドユーザーの直接対話を禁止することで，できあがるソフトウェアがエンドユーザーが期待していない見当違いのものになったり，それを修正するためにむだな時間と費用が発生する状況です．

▶**再構想解：**リスク主導開発，すなわちプロトタイピングとユーザーフィードバックをともなうスパイラル状開発スタイルを導入します．ユーザーフィードバックが重要です．単に開発スタイルをウォーターフォールからスパイラルに変えても，開発者をユーザーにふれさせずに暗室に閉じ込めているなら状況が全然変わらないからです．

ソフトウェアのアーキテクチャのアンチパターン(第6章)

ここで述べられているアンチパターンは，文字どおりソフトウェアのアーキテクチャ上に(つまりシステムの構造や構成に)見い出されるものです．ただし，アーキテクチャのレベルで問題をかかえているものは，個人はもちろんのこと一つの企業や組織レベルにまで到達する可能性が大きいものです．第5章で述べたものより関わる人員や対策すべき範囲が広がっているため，さらにやっかいな状況といえるでしょう．

● おんぼろ煙突化

既存のソフトウェアシステムをそのまま分散インフラストラクチャに移行させたため，従来どおりの設計が分散コンピューティングで不都合を起こしている状況です．たとえば，非分散環境で情報を転送するときの粒度が小さくてもとくに問題が起きないのが，分散環境では実行効率を落とすことがあります．

▶**再構想解：**インターフェースのリエンジニアリングを行います．既存のインターフェースで不都合な箇所を洗い出し，それを分散環境に適したインターフェースに置き換えます．

● 全社的おんぼろ煙突化

ソフトウェアの変更や拡張が行き当たりばったりで行われ，後になればなるほど変更や拡張が苦しくなり行き詰まってしまう状況です．それが一つの開発チームではなく全社的に行き渡っている場合，社内での再利用や相互運用でも障害となってしまいます．

▶**再構想解：**基盤的ないくつかのレベルにおける技術調整が必要です．システムの基本形に関する社内規格を確立し，これを使って全社的な基盤となる共通技術/共通製品の選定に関する調整を行います[注2]．

● 平直混在

水平的設計要素(複数のアプリケーションにまたがる設計要素)と垂直的設計要素(個々のアプリケーション特有の設計要素)が入り混じるために不安定なアーキテクチャを生じてしまう状況です．

▶**再構想解：**水平的設計要素を先に同定して，そこから垂直的設計要素を切り分けます．一つのアーキテクチャの中で水平成分，垂直成分，その他の成分を上手に均衡させます．

● システムのおんぼろ煙突化

複数のサブシステムの統合や連結が行き当たりばったりであったり共通するしくみがなくバラバラであるため，システムの拡張や変更がやりにくく不安定になってしまう状況です．

▶**再構想解：**ソフトウェアモジュールの柔軟な交換を可能にするためのコンポーネントアーキテクチャを確立させます．その際コンポーネントのインターフェースの適切な抽象化を見つけだすようにします．

● 総花主義

ドキュメント主導型ソフトウェア開発のため，ドキュメント記述者が重要な意思決定を避けたり安全第一主義に陥るあまり，役に立たなかったり焦点が定まっていない要件文書や仕様書を書いてしまい，それが開発の支障となってしまう状況です．

▶**再構想解：**ドキュメント作りに関して明確な目標と指針を与える必要があります．また，ドキュメントの品質を評価し正しくフィードバックさせるしくみも必要でしょう[注3]．

注2：参考文献1)では，この後，詳細な解決策にページを費やしていて，これを簡単に要約できない．ゆえに詳細は同書を参照していただきたい．

● 砂上の楼閣

特定ベンダや特定製品に依存することで，それらがアップグレードしたり複雑化してしまうと，とたんにソフトウェアの変更や相互運用の面で支障が生じ，絶え間ないメンテナンスや余計な出費に悩まされる状況です．

▶**再構想解：**これは「梯子外し」と同様，主導権を他者が握ってしまうことによる弊害です．「隔離層」と呼ぶ緩衝地帯を設けることで特定ベンダや特定製品の層と，他者の影響を受けたくない層を隔離してしまい，それぞれの層を交換可能にします．

● 羊頭狗肉

オープンであるとか標準規格を遵守していると称しながら実態がそうでない状況です．広く出回っていて標準規格のように錯覚しそうになりますが，それがいわゆるデファクトスタンダードの場合は要注意で，もともと明確な仕様書がなくバージョンの違いで頭を悩ませてしまうことがあります．

▶**再構想解：**これに関しては，技術問題よりもむしろ政治問題や政治レベルでの働きかけが必要です．標準規格を遵守していないベンダを告発するしくみ作りや効果的なメッセージを発することなどです．

● 文書化軽視

きちんとしたドキュメントのないまま(とくにアーキテクチャの仕様書がないまま)開発が行われ，結果的にプロジェクトの進行に支障が生じたり，メンテナンス性がきわめて低いソフトウェアが作成されてしまう状況です．

▶**再構想解：**システムのアーキテクチャを定義するための組織的な取り組みが必要です．システムに対する複数の視野を活用し，そのために「目標と質問によるアーキテクチャ」という手法を利用します．

● 無能集団化

開発プロジェクトの人数が多いものの，その大半が単に頭数をそろえるものであったり，そこにすわっているだけの無能者の集団と化していて，人数がいるわりにはちっとも仕事がはかどっていない状況です．

▶**再構想解：**ソフトウェア開発では，人数が多ければ多いほど各要員のスキルのばらつきがひどくなったり，コミュニケーションギャップによって期待したほどの効果がえられないことが多々あります．プロジェクトの理想的な人数は4人といわれています．なるべく少人数で機動的なプロジェクトを軸にした開発スタイルを検討すべきでしょう．

● 組織硬直

間違った「平等主義」によって開発にかかわる全員の意見を尊重したようでいて，じつは全員にとって役立たずの組織運営に陥っている状況です．できあがってくる設計や文書も過度に複雑で，それでいて一貫性がなく誰にも理解できない代物になってしまいます．

▶**再構想解：**解決の基本は会議過程の変革です．会議が効果的に運営する工夫がいろいろと必要になります．硬直化に陥らないために，ソフトウェアの開発工程における役割分担を明確にすることです．各人の役割が明確でないことで間違った平等主義に陥るからです．

● 万能ナイフ

異様に複雑なインターフェースをもったクラスが作られ，その複雑さゆえ支障が生じる状況です．このアンチパターンは「肥満児」と表面的に似ていますが，別物です．肥満児は一つのクラスに処理が集中するのに対し，万能ナイフは複数のクラスが複雑なインターフェースをかかえる状況です．つまり肥満ではなく重装備であったり過剰な装飾が行われている状態です．

▶**再構想解：**複雑な技術のどの部分をどのように使うかという規約を明記した「プロファイル」作りをします[注4]．

● 自作固執

すでに誰かが作成してくれたアーキテクチャやコンポーネントを再利用するのではなく，わざわざ自分で一から作り直す，いわゆる「車輪の再発明」にとらわれている状況です．

▶**再構想解：**「アーキテクチャ採掘」によって再利用や相互運用できるアーキテクチャやコンポーネントを見い出します[注5]．

● 実装溺愛

開発者が抽象化能力に欠けているため，必要以上に複雑な設計をしてしまい，開発はもちろんのことテストや文書化，メンテナンスなどあらゆる局面で面倒になっている状況です．

▶**再構想解：**残念なことに，きちんとした抽象化能力を有する設計者や開発者は小数派である[注6]ため，悪しき平等主義はこのアンチパターンを助長するだけです．ソフト開発においては役割分担を明確にし，きちんとした抽象化能力を有する者をアーキテクトなどの上流工程に固めるべきです．

プロジェクト管理のアンチパターン(第7章)

ここで述べられているアンチパターンは第5章や第6章のような「技術的」な問題ではなく業務の管理や遂行をする上で問題

注3：じつは参考文献1)では，総花主義の再構想解が見事に役に立たず焦点が定まっていない．したがって，この項目は筆者自身の独断で書いた．

注4：上記のように参考文献1)には書かれているが，複雑なものや過剰なものをありがたがったり高等なものだと勘違いする人間の生理的な誤謬をただしていくしくみ作りのほうが先決ではないかと思う．

注5：アーキテクチャ採掘の反対語が「アーキテクチャ耕作」で，これは従来どおりの要件主導型のアーキテクチャ設計で行われるもの．皮肉なことに，ほとんどのソフトウェア開発は従来どおりのセオリを守っているため，結果的に再利用や相互運用が進まない現場だらけになってしまう．

注6：参考文献1)にも挿話証拠として「良質な抽象を定義できる開発者は5人のうち1人しかいない」と聞いた現場のアーキテクトが「とんでもない．50人のうち1人だ」と反論した話があるが，実際問題，そんなものである．それどころか，欧米ほど論理能力や抽象化能力が鍛えられていない(?!)日本の場合は，さらに状況が悪いと実感する．ソフトウェアの要として「抽象化能力」は基本中の基本だが，これを教えたり伝えることは本当に困難をきわめる．

となってくるものです．いくつかのアンチパターンは，開発者よりも管理職や経営者の問題であり，それゆえ開発者レベルでは解決が不可能に近いというやっかいな性質をもっています．

● 口八丁手○丁 注7

知ったかぶりの「業界通」の意見などによって，これを真に受けた管理職や意思決定者からの質問や圧力で，本来の仕事に時間がさけず支障をきたす状況です．

▶再構想解：真偽をきちんと見分けられるエキスパートを社内で確保しておきます 注8．

● 分析地獄

設計のためにどこまで詳細な分析をするべきかの歯止めが効かず，どんどん深い分析に突入してしまい，なかなか次の段階に移行できない状況です 注9．

▶再構想解：このアンチパターンは，完全な分析をしようという間違った完璧主義に分析者がとらわれていると発生しやすいものです．皮肉なことにオブジェクト指向開発をしているのに古典的なウォータフォールモデル，すなわち，

(1) コーディング工程前に詳細分析が完了している前提
(2) あらゆる事象が事前にすべてわかっている前提
(3) 分析結果に対し，開発途中で拡張や修正がない前提

という，今となってはあり得ないことが知れわたっている前提に従おうとしたときに陥りやすい誤謬です．オブジェクト指向開発をする場合，ウォータフォールよりもスパイラル状開発スタイルを採用すべきです．

● 文書化地獄

開発者が説明用資料や企画文書の作成に追われてしまい，本来のソフトウェア開発に着手できない状況です．

▶再構想解：文書化地獄に陥っている開発者を救うためにプロトタイプ構築を命じます．プロトタイピングの活用は，スパイラル状開発の重要な要素ともなります．

● 計画倒れ

過剰で綿密かつ詳細な計画を立てるものの，それゆえに計画を遵守できない状況です．あるいは文書として見栄えが良いものの，根拠のない前提で計画が立てられたときにも陥るアンチパターンです．

▶再構想解：皮肉なことに誰にも守れないような緻密な計画を立てて，それが予定どおり（？）失敗しても，その原因は「緻密さが不足していたからだ」と勘違いしてしまう傾向があります．真の原因はそんなところにあるのではなく余裕のなさなのですが，それに加えて状況が刻々と変化するのが当たり前なのに 注10，その対応ができない運営方法だからです．そのためには漸進的な計画修正が含まれるべきです．また根拠のない前提で計画を立てるのではなく正当な根拠で計画を立てるべきです．ただし「正当な」とは，管理職や経営者がこのスケジュールで作ってほしいとか，誰かを安心させるためであってはならないのはいうまでもありませんが．

● 取り越し苦労

プロジェクトが成功しそうだというときに，ひょっとして失敗するのではないかと疑心暗鬼にとらわれる状況です 注11．

▶再構想解：プロジェクトの成功宣言を管理職がすることです．実際の成果が不確かであったとしても宣言は明確に行うべきです．開発プロジェクトの志気を高めることが大事だからです．

● 横紙破り

「横紙破り」や「文句屋」と呼ばれる問題行動を起こす人物がソフトウェア開発チームに悪影響をもたらす状況です．やっかいなことに，この手の「困ったちゃん」はそれなりに技術力があったり管理職であることもあり，簡単に切り捨てるわけにいかないので問題をややこしくすることがあります．しかし，何事にも限度というのがあるので放置は最悪の選択です 注12．

▶再構想解：このアンチパターンを解決するには，戦技/戦術/戦略の複数レベルで考えます．

(1) 戦技レベル

一つの手としては，文句を言う本人に解決するための全責任を与えて一定時間以内に解決させるのです．口先だけの場合は解決できなくて本人が恥をかき，居づらくなるでしょうし，解決すればそれで文句が納まるかもしれません．

(2) 戦術レベル

管理職が問題人物と個別に面談したり，人材スカウトに問題人物を紹介して本人を円満退社させるという手があります．

(3) 戦略レベル

文句屋を1箇所に集めて「共食い」をさせたり，窓際に追いやったり組織から追い出します．ただし，注意しないといけないのは窓際になって暇をもてあますことで，余計に困った行動に精力をついやす事態を防止しないといけません．

注7：訳書では「○（まる）」となっているが「0（ゼロ）」の可能性もある．というのも口先だけで手先がともなっていない状況を表現しているから．

注8：あくまでエキスパートであって「オタク」や「マニア」や「知ったかぶり」ではない．どこの国の軍隊でも参謀役はきちんとしたエキスパートの軍人が行うものであって，「軍事オタク」にまかせていないのと同様に．

注9：参考文献1)ではオブジェクト指向開発での古典的なアンチパターンと説明しているが，そんなことはない．それ以前から存在した悪しき状況である．ここの項目だけに限らないが同書はオブジェクト指向開発ばかりに偏っていて，それ以外でも同様なアンチパターンが生じていることにどういうわけかふれていないことがある．

注10：「戦争論」で有名なクラウゼビッツの表現にしたがえば「摩擦」に相当するもの．物事はすんなり行くほうが奇跡的状況であって，ほとんどは計画時点では想定しなかった事象によって日程が延びるものである．

注11：たまたまこの原稿を書いている時点で何年かぶりでリーグ優勝しそうだという某球団が話題になっているが，この球団の熱烈なファンたちが陥っているのが，このアンチパターンのような気がする．

注12：参考文献1)の中でもこの項目は最高に面白くタメになるので，筆者は個人的には，ここの項目だけでも本を買う値打ちがあると思う．とくにソフトウェア業界はこの種の「困ったちゃん」(?!)にあふれていて，実害をよく目撃する．

● 知的暴力

最新理論や難解な技術に通じていることを吹聴する人がいることで，そうでない人たちをびびらせたりチームの和を乱す状況です．

▶**再構想解：**これは一見「横紙破り」と似ていますが横紙破りは利己的な人物として排除すべきなのに対し，こちらはむしろ積極的に取り込むことで組織の利益に転じやすい点が違います．知的暴力が生じないように全社的に教育訓練の機会が与えられるようにし，レベルの高い人たちがそうでない人たちに親切に教えられる環境作りが大事です．

● 理性の欠落

管理者や意思決定者の性格上の問題によって，まともな意思決定ができずプロジェクトに支障をきたす状況です．管理者の管理能力の欠落，あるいはやたらに重箱の隅つつきにこだわり大局観が欠落している場合にも見い出されます．

▶**再構想解：**次のガイドラインにしたがい，問題を一つ一つ解決していきます[注13]．

① 管理者自身が問題を自覚し助けを求められるようにする
② 管理者は開発スタッフを理解する必要がある
③ 管理者は明確な短期的目標をスタッフに提供する必要がある
④ 管理者とスタッフが目的を共有し，同じ目標に向かってまい進できるようにする
⑤ 工程の改善可能箇所を探す
⑥ 互いのコミュニケーションを促進する
⑦ コミュニケーションのメカニズムを管理する．掲示板やメールなどはバトルを生じやすいので注意する
⑧ 過剰管理に陥らず，例外的現象のみを管理する
⑨ 効果的な意思決定テクニックを適用する

● 誇大営業

エンドユーザーに本番ではなくデモソフトを見せてしまったことで，それが本番のソフトであると勘違いさせ，実際の製品がデモと似ても似つかない低品質なものであったり，製品化できないなどでエンドユーザーを失望させてしまう状況です．

▶**再構想解：**正しいエンドユーザーの「期待管理」を行います．つまり過剰の期待を抱かせるのではなく，実際の製品レベル以下を期待させるようにします．実際の製品を目にすると，それが期待以上であるためユーザーを驚かせて喜ばすのですが，これが逆だとユーザーが離反してしまうのはいうまでもありません．

● 暴走プロジェクト

プロジェクトの管理がまともになされていない状況です．アーキテクチャ面での戦略が欠けていて実装しづらい設計になっていたり，コードの試験やレビューがおざなりになっていたり，試験が粗雑であるなど，よくこれでプロジェクトが動いているものだ（実際には動いていないが）と驚きあきれられてしまう状態です．

▶**再構想解：**正しいリスク管理によって暴走プロジェクトが起きないよう手を打ちます．リスク管理にはいくつかの視点があります[注14]．

● あとは野となれ

コードはできあがっているもののドキュメントがないか，あっても意味があるドキュメントではなく形骸化しているため，本当に欲しいドキュメントが用意されていない状況です．

▶**再構想解：**きちんとしたドキュメント作りができるよう教育訓練をします．またドキュメントの形骸化が起きないように柔軟なガイドラインを決めるなど，管理職レベルの支援も必要になるでしょう．

● 泥縄

前段階の工程（計画/検討/分析）にやたら時間がかかってしまったため，後の段階（設計/実装/試験など）を短期間で片付けざるを得なくなる状況です．全工程のスケジュールがまんべんになっておらず，前半はやたら暇をもて余し，後半はデスマーチ[注15]になってしまう，典型的なダメプロジェクトが陥りがちなアンチパターンです．

▶**再構想解：**開発スタッフの志気を著しくそぎ落としてしまう泥縄型開発スタイルを脱却するために，アーキテクチャ主導型開発スタイルに移行します．ただしアーキテクチャ主導型は長期的な視野と取り組みを要求します．泥縄スタイルが身に付いた組織がそこに移行するのはかなりの努力を要すると思われます．

● お家騒動

管理者同士の確執や抗争によって組織が迷惑をこうむる状況です．

▶**再構想解：**「ピザパーティ」テクニックなどを使って，当事者同士の意見調整や和解にもち込みます．

● 人食いメール沼

メールを使った議論がこじれてしまい（1対1のメールだけでなくメーリングリスト，掲示板，チャットなども含むだろう）メンバ同士の不和や志気低下に陥る状況です．

▶**再構想解：**メールの便利な側面だけでなく危険な面にも注意し，微妙な問題に関してはメールを避けて直接対話したり，メールを使ったバトルに進展しないよう注意します．

参考文献

1) William J.Brown, Raphael C. Malveau, HaysW.“ Skip ”McCormickIII, Thomas J.Mowbray 共著，岩谷宏訳，『アンチパターン――ソフトウェア危篤患者の救出』，ソフトバンク

みやさか・でんと　miyadent@anet.ne.jp

注13：参考文献1）では，詳細な解決策にページを費やしていて簡単に要約できない．ゆえに詳細は同書を参照していただきたい．
注14：参考文献1）では，視点にページを費やしていて簡単な要約はできない．詳細は同書を参照していただきたい．
注15：「死の行進」とも呼ぶ．多くの犠牲と損失や高い確率で失敗が予測されるプロジェクトを示す自虐的な表現．

「VxWORKS」を使ったRTOS技術の基礎と応用

第2回

ネットワークプログラミング――Webサーバ編

高山 剛

メディアで「ユビキタス」という文字を見ない日はない今日この頃ですが，その基礎となるネットワーク技術，なかでもTCP/IPは，エンジニアにとっては必須の技術です．TCP/IPはインターネットが社会的革命を起こす前から存在し，組み込みの世界でも使われていました．というより，組み込みの世界のほうが将来性を期待されていたようにさえ思います．

たとえば，工場にある装置をインターネットで接続して受注から生産へと一括管理したり，プラントシステムでは遠隔地のセンサからのデータ収集や装置の操作，ビルの管理，車載のネットワーク，ネットワーク化による多重化システムなど，ネットワークの応用は必然のものでした．**図1**のように，リアルタイム性が要求されるアプリケーションやミッションクリティカルな処理，パフォーマンスが要求される処理をRTOSであるVxWORKSが担当し，GUIやデータベースなどのリアルタイム性の要求されないアプリケーションに関してはUNIXを利用し，二つをネットワークで結ぶというアプローチを，TCP/IPの登場当時からWIND RIVERは提供してきました．

同時に，ネットワークの高速性，UNIXで普及したFTP，NFS，TELNETなどのネットワークアプリケーションを全面的に活用した開発環境も提供してきました．ネットワークを応用した開発環境が優れていたため，実際のアプリケーションがネットワーク機能を必要としなくても，アプリケーション開発時やメンテナンスの目的でネットワークは組み込みエンジニアに利用されてきました．インターネット全盛の現在では，情報家電がネットワーク機能を必要とするのは当然ですが，ネットワーク技術が激烈な競争の中で，Gbpsを越える伝送速度のEthernetや，スイッチング技術が低価格で入手可能となり，まったく新たな応用が登場しています．

〔図1〕VxWORKSとUNIXとネットワーク

たとえば，PICMG2.16と呼ばれる次世代PCIバスでは，PCIバスの転送速度を上げるためにクロックを上げようとするとPCIバスのスロット数が減るというジレンマがあり，それを乗り越えるため，Ethernetの接続配線をPCIバスの一部に配線し，バスインターフェースとしてEthernetチップを利用し，さらに各CPU間の通信がGbpsの通信が保証されるように，バスにIPスイッチング機能が採用されています．このようにバスアーキテクチャにネットワーク技術を採用したときは必然的に，ネットワークプロトコルスタックをもつOSが必要となります．従来はバス上の共有メモリによりOSがなくても開発が可能だったかもしれませんが，今後はネットワーク対応OSが必須となるでしょう．今後の組み込み系ソフトウェア技術者に必須なのは，コンピュータ言語，OSの技術，そしてネットワーク技術です．

今回は，HTTP（Hyper Text Transfer Protocol）サーバ（Webサーバとも呼ばれる）を例に，VxWORKSでのネットワークプログラミングの実際を紹介します．組み込みで初めてEthernetなどのネットワークを応用しようと考えている方や，VxWORKSのネットワークのプログラミングやデバッグ環境をまだご存知でない方，組み込み機器にWebサーバを組み込んでみようと考えている方に役立てばと思います．Socketインターフェースなので，VxWORKS以外のRTOSでも使用できると思います．

WWWの根本原理について

WWW（World Wide Web）のしくみは驚くほど簡単です．WebブラウザはURLで指定されたホストに，HTTPを使ってHTMLファイルを要求し，ページ記述言語であるHTMLをブラウザが解析，HTMLに含まれているタグでイメージファイルや音声ファイル，スクリプトが指定されていればHTTPでさらにそのファイルを要求し，一つのページとして表示します．HTMLがさらに他のURLを含んでいたときにブラウザでクリッ

クされれば次のURLへアクセスを行います．これがハイパーリンクと呼ばれるしくみです．世界のWebサーバがハイパーリンクで簡単にリンクできたことが，WWWが成功した理由でもあります．

HTTPを解析する

ブラウザとWebサーバ間での通信に用いられるHTTPのプロトコルをご存知でしょうか？ WWWと同様，HTTPは非常に単純です．たとえばInternet Explorerから`http://yourUNIX/books.html`としてUNIXでsnoopコマンドなどでモニタすると，次のようにHTTPプロトコルを覗き見できます．

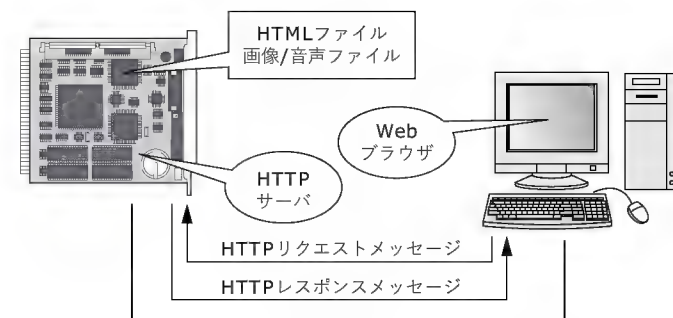
```
GET /books.html HTTP/1.1
Accept: */*
Accept-Language: ja
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible;
MSIE 6.0; Windows NT 5.0; .NET CLR 1.0.3705;
                        .NET CLR 1.1.4322)
Host: 147.11.60.121
Connection: Keep-Alive
```

これは，HTTPリクエストメッセージと呼ばれるもので，とくに注意してもらいたいのは最初の1行目（リクエスト行と呼ばれる）の`GET /books.html`というリクエストです．HTTPサーバはこのリクエスト行に対してHTTPレスポンスメッセージを返します．HTTPサーバは前述の例では`GET /books.html`のリクエストに対して`/books.html`ファイルの中身をレスポンスメッセージの中身として応答することになります．ブラウザとHTTPサーバとは基本的に，このようにリクエストとレスポンスメッセージの二つの繰り返しで成り立ちます（**図2**．WWWの発達した現在では，毎回TCP/IPコネクションを張るこのような方式は非効率なため，1回のコネクションで複数のファイルのやり取りを行うように拡張されている．もちろん過去への互換性も同時に保っているため，過去の方式も許されている）．このHTTPリクエストメッセージは，リクエスト行，ヘッダ，メッセージ本体というように大きく三つのフィールドに分かれ，さらにリクエスト行は，

メソッド　リクエストURL　HTTPバージョン

という構文になっています．ヘッダはブラウザの受け入れられるアトリビュート（属性）を示しています．今回のTiny HTTPDでは，ヘッダは考慮せず無視します．興味のある方は，HTTP1.1の仕様書であるRFC2616を見ると良いでしょう．またメソッドも，HTTP1.1にはGETのほか，PUT，POSTなどメソッドがありますが対応していません．必要があれば対応してみてください．

リスト1（稿末：pp.178-179）のTiny HTTPDサーバは，ネットワークアプリケーションの一例として作成し，Windowsの

〔図2〕HTTPのしくみ

Internet Explorerとの間で動作確認したものです（とりあえず動いている程度のものでHTTP1.1に対応しているとはいえないので理解いただきたい）．このプログラムは，ポート80（HTTPサーバのWell-known-port）であらゆるホストからのHTTPリクエストを受け付け，レスポンスメッセージを返信します．また，VxWORKSの機能を生かして拡張機能ももたせています．

実際にVxWORKSで動かしてみよう

まず実際に動かすために，**リスト1**の`httpd.c`をコンパイルします（ここではTornadoのGUIを使用すると直感的で簡単だが，説明する分には図を多用しないとわかりにくくなるので，コマンドラインの方法で説明する）．VxWORKSでアプリケーションをコンパイルするには，推奨されているオプションやヘッダのディレクトリパスを指定する必要があり面倒ですが，BSPのディレクトリで`make`コマンドを使うと便利です．この例の場合，

```
make httpd.o
```

とします．これはBSPのディレクトリに含まれるMakefileで`xxx.o`が与えられた場合のルールが定義されているためです．

これで，コンパイルが終わりました．次に，ターゲットにプログラムをダウンロードしなくてはなりません．他のOSではカーネルとリンクしなければならない場合があり，時間がかかるうえにターゲットをいちいちリセットしなくてはなりません．しかしVxWORKSでは，

```
ld < httpd.o
```

でダウンロードできます．ここで，

```
Errors while downloading D:/T221pen/
   target/proj/Project1/PENTIUM3gnu/httpd.o:
symFindByNameAndType
```

となることがあります．VxWORKSのデフォルトではINCLUDE_NET_SYM_TBL機能などが足りていないので，VxWORKSを再コンフィグレーションしてこれらの機能を組み込んでください（くわしくはマニュアルを参照してほしい）．

エラーが出なければ，ダウンロードが完了しています．逆ア

〔図3〕逆アセンブラで確認する

```
-> l httpdInit
httpdInit
    00bc0170  55                    PUSH        EBP
    00bc0171  89 e5                 MOV         EBP, ESP
    00bc0173  83 ec 18              SUB         ESP, 24
    00bc0176  83 7d 08 00           CMP         [EBP+8], 0
    00bc017a  75 07                 JNE         httpdInit + 0x13
    00bc017c  c7 45 08 50 00 00 00  MOV         [EBP+8], 0x50
    00bc0183  e8 fc ff ff ff        CALL        httpdInit + 0x14
    00bc0188  83 ec 04              SUB         ESP, 4
    00bc018b  6a 00                 PUSH        0
    00bc018d  6a 00                 PUSH        0
```

〔図4〕i()コマンドの実行

```
-> i
  NAME        ENTRY       TID    PRI   STATUS      PC       SP     ERRNO  DELAY
---------- ------------ -------- --- ---------- -------- -------- ------- -----
tExcTask   excTask      7fb1230   0 PEND         3d8f23  7fb1178       0     0
tLogTask   logTask      7fae890   0 PEND         3d8f23  7fae7c8       0     0
tShell     shell        7f2c1a0   1 PEND         319eeb  7f2be04       0     0
tWdbTask   wdbTask      7f2d3c4   3 READY        319eeb  7f2d2b8       0     0
tNetTask   netTask      7f4dfa4  50 READY        319e51  7f4dde8       0     0
httpd      0xbc1380     7f243f8 100 PEND         319eeb  7f240cc  880226     0
tDcacheUpd dcacheUpd    7f8d0bc 250 DELAY        3c9d51  7f8d044       0     5

value = 0 = 0x0
->
```

〔図5〕Webブラウザから送信されてきたリクエストの表示

```
GET / HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-
excel, application/vnd.ms-powerpoint, application/msword, application/x-shockwav
e-flash, application/pdf, */*
Accept-Language: ja
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET CLR 1.0.3705
; .NET CLR 1.1.4322)
Host: 147.11.60.121
Connection: Keep-Alive

Open home.htmlcan't open error home.html
```

センブラで確認してみると，**図3**のようになっています．

実行できる環境は整ったので，Entryルーチンである httpdInit()を呼び出します．

```
->httpdInit
```

VxWORKSではこのように，あらゆる関数を直接呼び出すことができます．ちなみに，このプログラムのグローバル変数 httpdStatusを入力すると，

```
->httpdStatus
httpdState = 0xbc0a2c: value = 0 = 0x0
```

とグローバル変数を参照することができます．

httpdInit()は，httpdタスクを起動します．

ここでi()コマンドを実行すると，全タスクを参照できます(**図4**)．

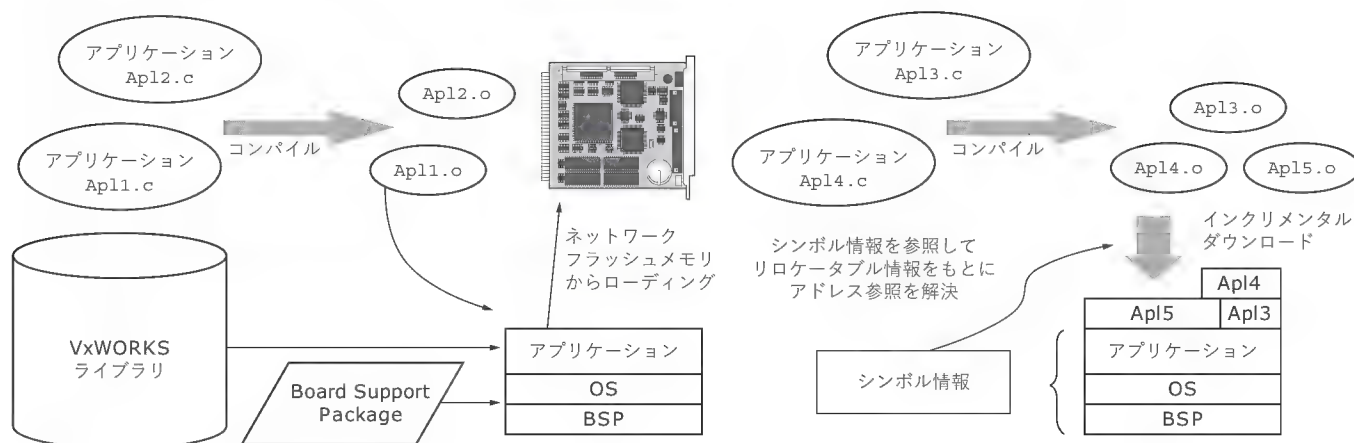
ほかにも数々のコマンドが用意されています．help()コマンドで確認してください．一つユニークなのは，tt()コマンドです．tt()コマンドは，指定したタスクのスタックをバックトレースし，現在実行中の関数が，どの関数から呼び出されたかをバックトレースしていき，可能ならば(アーキテクチャによる)引き数も表示します．たとえば，このTiny HTTPDであれば，

```
-> tt httpd
31c9aa   vxTaskEntry    +a   : bc143f
              (50, 0, 0, 0, 0, 0, 0, 0, 0, 0)
bc14c3   httpDaemon     +113: accept
              (5, 7f243a4, 7f243a0, eeeeeeee)
3605ec   accept         +6c  : bsdAccept
            ([5, 7f243a4, 7f243a0, 3ad7c8, 0])
34a497   bsdAccept      +97  : semQPut
               (7f29760, 0, 7f24138, 34a430)
value = 0 = 0x0
```

とすることで，現在どの関数が実行中で，どの関数から呼び出されたかを知ることが可能です．ここでは，SocketのAPIであるaccept()からTCP/IPプロトコルスタックを経てbsdAcceptが呼び出され，semQPut(セマフォの内部関数)関数内でペンドしていることがわかります．

Webブラウザからアクセスしてみる

では，実際にWebブラウザからアクセスしてみましょう．

〔図6〕インクリメンタルな開発

ブラウザ(Internet Explorer)から,

```
http://targetのIPアドレス
```

を指定してください．ターゲット側のコンソールに，Webブラウザから送信されてきたリクエストが表示されます(**図5**).

これは，デバッグ目的で表示しています．コード中に，

```
if ( httpdDebug ) printf ( "¥n %s ", buf );
```

の行があり，グローバル変数httpdDebugが=1となっているため，デバッグ目的にリクエスト内容が表示されるわけです．このデバッグ表示が邪魔になったとき，当然このグローバル変数httpdDebugが=0であればと思うでしょう．VxWORKSでは，

```
>httpdDebug=0
```

とすることで，自由に任意のタイミングでグローバル変数を変更できます．VxWORKSは，プログラム中に含まれる関数やグローバル変数，スタティック変数についてシンボルとその属性をシンボルテーブルで保持しているため，グローバル変数は，特定アドレスからのデータの参照，関数であれば関数コールするように自動的に解釈し実行しています．コードの最後にhttpdShow()というどこからも呼び出されない関数がありますが，これをSHELLから呼び出すことも可能です．

```
-> httpdShow
THTTPD statistics
Get Request     6
Not Found        1
Unknown Request 1
Total of transfer size 3320
```

この関数は，単にプログラム中で使用されるグローバル変数や構造体を16進数で表示するだけですが，もっと高度で複雑なプログラムにこのようなデバッグ用の関数をあらかじめ組み込んでおき，複雑なデータ構造を表示したり，特殊なタイミングでデータを表示したりといった応用も可能です．

最初に，ld < httpd.oと，動的にアプリケーションモジュールをダウンロードしましたが，同様にいくつでもインクリメンタルにダウンロードが可能です(**図6**)．これを利用すると，ときにたいへん強力なデバッグ機能になります．たとえば，あるとき，1秒単位での通信量が見たいと思って**リスト2**のようなアプリケーションを作ってコンパイラを呼び出すと，コンパイルに1秒，さらに1秒でダウンロードができます．httpdStatはexternで宣言されているので，もともと存在したhttpdStatのアドレスを参照するようローダがアドレス解決します．httpdPriodShowを呼び出すと，httpdのデータ構造を参照していることに注目してください．

〔リスト2〕1秒単位での通信量を見る

```
extern struct httpdStat
    {
    int NoGetRequest;
    int NoNotFound;
    int NoBytes;
    int NoUnknownRequest;
    } httpdStat;

httpdPriodShow()
    {
    int old= httpdStat.NoBytes;
         for (;;)
         {
    printf ( "Delta/sec %x¥n" , httpdStat.NoBytes -old);
    old = httpdStat.NoBytes
    taskDelay(60);  /* 1秒遅延 */
         }
    }
```

```
>httpdPriodShow
Delta/sec 0
Delta/sec 0
Delta/sec 23340
```

このようにターゲットシステムの動きを止めることなく，モニタできてしまいます．さらに，1秒単位でなく10秒単位でも見たいし，30秒単位でも見たい場合は，

```
httpdPriodShow(int arg)
     {
```

```
        int old= httpdStat.NoBytes;
                for (;;)
                {
        printf ( "Delta/sec %x¥n" ,
                        httpdStat.NoBytes -old);
        old = httpdStat.NoBytes
        taskDelay(60 * arg);  /* N秒遅延 */
                }
        }
```

とし，コンパイル・ダウンロードし，同様に，

```
>httpdPriodShow 10
```

とすると，10秒単位で通信量のモニタができてしまいます．制御システムのある先生が「測定できなければ，制御できない」と言ったそうですが，VxWORKSの場合は，思いたったらコードを継ぎ足して，直に呼び出せるので，思考停止せず，システムのふるまいをモニタできるのがうれしいところです．for(;;)で無限ループではないか？と思われかもしれません．たしかに教科書的には，終了条件を決めてプログラムを書けばよいのですが，VxWORKSのようなOS下ではCTRL-Cで強制的にシェルをリスタートして強制終了してしまいましょう．シェルをリスタートしてバックグラウンドで動作させるなら，

```
>sp httpdPriodShow,10
```

とすれば，新しいタスクとしてhttpdPriodShowをバックグラウンドで動作させることができるでしょう．httpdShow()のように，そのアプリケーションしか知らない内部データ構造などの表示機能があれば非常に便利なのはおわかりと思います．VxWORKSのミドルウェア，ドライバ，プロトコルスタックは，可能な限りこのようなデバッグ用関数をもっています．この種類の機能はすべてxxxxShow()という名付けの規則をも持っているので，現在のVxWORKSが一体どのようなShowルーチンをもっているかを**図7**のようにして調べられます．

さて，まだターゲット側でファイルシステムを用意しておらず，返信すべきHTMLファイルがないので，ブラウザにはファイル未検出というメッセージしか表示されません．ここで，カレントディレクトを移動し，HTMLファイルをもつディレクトに移動します．VxWORKSはデフォルトでリモートファイルシステムをもっているので，リモートファイルシステムの特定のディレクトリに移動します．

```
>cd "host:/tornado/docs"
```

ここで，なぜ""が必要かというと，cdは一見，コマンドのように見えますが，先ほどの関数と同じく単なる一つの関数で実装されています．そして引き数として，文字列のポイントを引き数として取ります．なので文字列の始端と終端(NULL)が特定できるようにするために，""で区切ることになっています．確認のため，

```
>ls
Books.html
 ⋮
```

とするとと，Books.htmlが存在するので，ブラウザでhttp://target ip address/Books.htmlとすると，Books.htmlの内容がブラウザに表示されるはずです．

〔図7〕現在のVxWORKSがどんなShowルーチンをもっているか調べる

```
>lkup "Show"
envShow                     0x0038bf50 text     (vxWorks)
pciConfigFuncShow           0x00310f50 text     (vxWorks)
symShowInit                 0x003977c0 text     (vxWorks)
pciConfigTopoShow           0x00311870 text     (vxWorks)
tcpstatShow                 0x00363120 text     (vxWorks)
excShowInit                 0x00316540 text     (vxWorks)
vxDrShow                    0x0031bfc0 text     (vxWorks)
routestatShow               0x0035e600 text     (vxWorks)
eventRsrcShow               0x003ccb80 text     (vxWorks)
dosFsShow                   0x00380180 text     (vxWorks)
<略>
```

世界を変えたHTMLとHTTP

HTTPはこのようにじつに簡単です．WWWではHTMLで実現され，ブラウザはHTMLにしたがい，必要があれば画像データ音楽データなどをHTTPにより取得します．たいへん簡単なしくみで誰でも考え出せそうですが，複雑にせず単純に抑え，現実社会の情報社会の需要にじつによくマッチしたのが成功のもとでしょう．逆にインターネット社会では，コンピュータ内でたとえどんなに優れたしくみでも，社会にマッチしなければ成功しません．組み込みの分野とインターネットの融合はこれからですが，WWWのように人間社会や経済まで変える技術が生まれてくると信じています．情報家電とブロードバンドの普及の進む日本にはその土壌があるのではないでしょうか？

さてインターネットで大成功したWWWですが，組み込みでの応用を考えてみましょう．たとえばプリンタを例に取ると，インクカートリッジの取り替え方はどうしたらいいのか？と途方にくれる場合がありますが，ネットワークプリンタにHTTPサーバを載せてドキュメントを用意し，各人のPCからドキュメントが参照できれば便利でしょう．紙詰まりの場合は，センサ情報を集めてHTMLで返信すれば，プリンタの紙詰まりの原因をグラフィカルに知ることができます(**図8**)．スプールの情報や課金情報なども見られます．文字情報や画像情報だけでなく，音声やJavaを用いた動画やインタラクティブな操作もできるでしょう．工場での産業用コンピュータでは，高価なGUIを廃止し，必要な際にPDAからWebでアクセスして設定を変えたり，モニタリングができ，実際に採用事例が増えているようです．

Socketプログラミングについては簡単に

ここでは，あえてSocketプログラミングについてくわしく解説しません．このプログラムはHTTPサーバではありますが，

もっとも単純なSocketプログラミングのサーバ側プログラムです．なので，これ以上簡単なものはないといっても過言ではありません．しかし，何の参考文献もなしにUNIXやVxWORKSのAPIだけ眺めても，コードは書けないのも事実です．ぜひコピーしてあなたのプログラムに使ってください．Socketプログラミングとはそのようなものです．筆者もネットワークプログラムを作成する場合は，昔書いたコードもベースにして，新しいコードを書いていきます．昔書いたコードというのは，自分のノウハウのデータベースといえます．このTiny HTTPDをベースにぜひ，自分でこのプログラムを拡張し，他のSocket APIの動作実験もしてみて，いろいろな設定や機能を実験して自分のデータベースとして活用すると良いでしょう．クライアント側のコードは，次のURLから見ることができるようにします．

http://www.windriver.com/japan/news/article/index.html

UNIX上で，HTTPのクライアントとして動作するはずです[注]．

Tiny HTTPDの独自拡張について

UNIX/Windowsの世界では，Webサーバに対して動的な拡張を施す場合，CGIとしてJava，Perlなどのインタプリタ言語で記述する方法が好まれ，メンテナンス性の面で採用されているようです．しかし組み込みでは，Webサーバに動的拡張性をもたせるためにインタプリタを搭載するには，メモリ量的にもパフォーマンス的にも機能的にもオーバスペックになってしまいます．組み込み機器では前述のプリンタのようにHTTPDを搭載してもマニュアルや紙詰まり情報など提供する情報がかなり限定され，さほど拡張性やメンテナンス性を要求されないので，インタプリタを使わず，C言語で記述され用意されたインターフェースをWebサーバから直接呼び出すほうが現実的と思われます．

Tiny HTTPDでは，URLで`/VxCMD?function`，たとえば`http://90.0.0.2/VxCMD?memShow`が指定されると，その関数（ここではmemShow）のエントリアドレスをシンボルテーブルから検索し，呼び出しを行い，その標準出力をHTTPレスポンスとしてとして返します．VxWORKSのI/OシステムはUNIXの流れを組んでいるので，`ioGlobalStdGet/Set()`で標準入出力を切り替える（リダイレクト可能）ことができます．UNIX同様にソケットディスクリプタは，ファイルディスクリプタと同様に`open()`，`read()`，`write()`関数で扱えるので，このような扱いができます．

`printf()`でプレーンのテキストを出力したり，HTMLで多少の装飾を加えることも可能です．現在の実装では引き数までは処理していませんが，必要であれば，ぜひ拡張してください．この拡張機能はセキュリティを考えていないので，たとえばRebootも直接のメモリへの変更も何でもできます．許されたAPIだけが許されるように，セキュリティの変更が必要になります．

〔図8〕組み込みシステムへのHTTPサーバの応用

このようにセキュリティは，HTTPDではたいへん重要です．このプログラムでは，HTTPリクエストメッセージに対して固定長を割り当て，それ以上のメッセージ長は受け入れないコードになっていますが，セキュリティホールを作らないようにバッファーオーバフローには常に神経を尖らす必要があります．アクセスを許すファイルの制限なども必要です．また，HTTPDからメッセージをIDやパスワードなどのメッセージを送る場合や，覗き見を防ぐためのセキュリティが必要になるでしょう．

WWWの世界はたいへんな勢いで発展しているので，新しいプロトコルやセキュリティを実現するのはたいへんです．パフォーマンスにおいても，同時に複数のHTTPリクエストメッセージへの対応やキャッシュ，ロギング，URLエイリアス，イメージファイル圧縮，文字入力，文字エンコーディング，リードオンリのROMファイルシステム，JavaScriptなどが必要になります．そのためHTTPDでも市販のミドルウェアが結局は使われることになります．WIND RIVERからは，プラットホーム製品の中にWIND MANAGE Web serverという製品をバンドルし，追加ロイヤリティなしに製品に使用できます．商品化を考えHTTP1.1への準拠が必須の場合，GUIのHTMLが複雑化して開発効率に支障を来たした場合など，より品質の高いミドルウェアを選択すべきだと思います．

たかやま・たけし
takeshi.takayama@windriver.com　ウインドリバー(株)

注：公開するコードに関して，使用制限はないが，不具合があっても筆者はいかなる責任も負わない．また，1個人によりコーディングし公開したものなので，WIND RIVER社とは関係なく，いかなる質問にも答えられないことをご理解いただきたい．

〔リスト1〕TINY HTTPDサーバ

```
/* httpd.c - Tiny HyperText Transfer Protocol Daemon */


/*
SYNOPSIS
This module provides WWW server as known as HTTP daemon.

DESCRIPTION
This module is very small module, however it provide great
capability for embedded devices. Imagine a copy machine connects
the Internet with a httpd task and html files in ROM disk.
It can provide its online manual and status changing every
seconds via WWW browser, in addition everyone can manipulate
the device by remote.

<How to use httpd>

vxWorks-> ld < httpd.o
vxWorks-> cd "www"          ..>>> html storage
vxWorks-> httpdInit

WWW browser --> http://your target name or ip address/
(eg. http://t120/books.html or http://147.11.60.121/index.html )

Author
Takeshi Takayama

This software is provided freely without any express or implied
warranty.  The authors will not be held liable for any damages
arising from the use of this software.
Permission is granted to anyone to use this software
                                                for any purpose.

*/

#include "vxWorks.h"
#include "types.h"
#include "inetLib.h"
#include "socket.h"
#include "ioLib.h"
#include "stdio.h"
#include "string.h"
#include "taskLib.h"
#include "symLib.h"
#include "ctype.h"

#define DEF_SOCK_PORT 80
#define MAX_HTTP_REQ   500
#define MAX_HTTP_SEND_BUF   5000
IMPORT SYMTAB *sysSymTbl;
extern void httpDaemon( unsigned short );
LOCAL httpdService (int sock,char *buf);
LOCAL httpdSendFile (int sock,char * file);

int httpdDebug = 01;
int httpdState;
struct httpdStat
    {
    int NoGetRequest;
    int NoNotFound;
    int NoBytes;
    int NoUnknownRequest;
    } httpdStat = { 0 , 0 , 0 , 0 };

/****************************************************************
*
* httpdInit  -  HTTPd Initialize
*
* This routine spawn httpd as a task.
*
*/

httpdInit( port )
    int port; /* port (default=80) */
    {
    int  rtn;

    if ( port == 0 ) port = DEF_SOCK_PORT;

    taskSpawn( "httpd",100,0,5000+ MAX_HTTP_REQ*2
                                            + MAX_HTTP_SEND_BUF,
    (FUNCPTR) httpDaemon, port,0,0,0,0,0,0,0,0,0);
    }

/****************************************************************
*
* httpDaemon  -  httpDaemon
*
* RETURN: none
*/

void httpDaemon (
    unsigned short port     /* port number */
    )
    {
    struct sockaddr_in sockaddr,ac_ad;
    int  length;
    int  msgsock;
    int   rval;
    int  i,m=1;
    int  optval;
    char buf[MAX_HTTP_REQ];
    int  s;

    s = socket (AF_INET, SOCK_STREAM, IPPROTO_TCP );

    httpdState = 1;

    if (s < 0)
    {
    perror ("socket error:");
    return;
    }

    bzero ((void *)&sockaddr, sizeof ( sockaddr) );
    sockaddr.sin_family = PF_INET;
    sockaddr.sin_port    = htons ( port );
    sockaddr.sin_addr.s_addr = INADDR_ANY;  /* ANY HOST */


    if (bind (s, &sockaddr, sizeof ( sockaddr ) ) == ERROR)
    {
    printf ("bind error:");
    return;
    }

    length = sizeof ( sockaddr );

    /* start accepting connections. */
    listen( s , 5 );

    do {
    bzero ( (void *) &ac_ad , sizeof sockaddr);
    msgsock = accept ( s,&ac_ad,&length );
                                /* client's name is uninterest */

    if ( msgsock == ERROR )
        perror ( "accept");
    else
        {
        if ((rval = read (msgsock ,buf , MAX_HTTP_REQ )) < 0 )
         {
         perror ("reading stream message");
         }
        else
         {
         buf[rval] = (char ) NULL;
         httpdService ( msgsock, buf );
         }

        }
        }
    while ( httpdState );

    close (s);
    }

LOCAL httpdService (
    int sock,
    char *buf
    )
    {
    if ( httpdDebug ) printf ( "%s¥n", buf ,0 );
```

〔リスト1〕TINY HTTPDサーバ(つづき)

```
    if ( ! strncmp ( "GET",buf,3  ) )
    {
    httpdStat.NoGetRequest ++;
    httpdSendFile ( sock ,buf + strlen ( "GET ")  );
    close ( sock );
    }
    else
    httpdStat.NoUnknownRequest ++;
    }

LOCAL httpdSendFile (
    int  sock,
    char * file
    )
    {
    int rtn,rtn2;
    char buf[MAX_HTTP_SEND_BUF+1];
    int fp;
    int i;
    char *html;

    for ( i=0;;i++ )
    {
    if ( file[i] == '¥n' || file[i] == (char ) NULL ||
             file[i] == ' ' || file[i] == 0x0d )
        {
        file[i] = (char ) NULL; break;
        }
    }

    if ( file[0] == '/' &&  file[1] == (char ) NULL )
    html = "home.html";
    else
    html = file+1;


    /* Special handling */
    if ( ! strncmp ( "VxCMD?",html,6 ) )
    {
    int old;
    FUNCPTR pAdrs;
    SYM_TYPE pType;
    int  er;
    char cmd[MAX_HTTP_REQ];

    old = ioGlobalStdGet(STD_OUT );
    ioGlobalStdSet (STD_OUT, sock );   /* stdout */
    printf ( "HTTP/1.1 200 OK¥n" );
    printf ( "Content-Type: text/plain¥n¥n" );

    sprintf ( cmd , "%s" , html+6 );

    er= symFindByNameAndType (sysSymTbl,cmd,
         (char **)&pAdrs,&pType
         ,SYM_TEXT | SYM_GLOBAL
         ,SYM_MASK_ALL);

    if( er == OK )
        pAdrs();
    else
        httpdFileNotFound( sock );


    ioGlobalStdSet (STD_OUT, old );
    return ;
    }


    if ( httpdDebug ) printf ( "¥nOpen %s" , html );
    fp = open ( html , O_RDONLY,0 );
    if ( (int ) fp == ERROR )
    {
    if ( httpdDebug ) printf ( "can't open error %s¥n",html );
    httpdFileNotFound( sock );
    httpdStat.NoNotFound ++;
    return;
    }


    for ( ;; )
    {
    rtn = read ( fp, buf , sizeof ( buf ) );
    if  ( rtn <= 0 )
        {
        close( fp );
        return;
        }
    httpdStat.NoBytes += rtn;
    rtn2 = write ( sock , buf , rtn );

    }
    }

httpdFileNotFound(
    int  sock
    )
    {
    int old;

    old = ioGlobalStdGet(STD_OUT );
    ioGlobalStdSet (STD_OUT, sock );   /* stdout */

    printf ( "HTTP/1.1 404 NotFound¥n¥n" );

    ioGlobalStdSet (STD_OUT, old );
    return ;

    }

httpdShow()
    {
    printf ( "THTTPD statistics ¥n" );
    printf ( "Get Request    %x¥n" , httpdStat.NoGetRequest );
    printf ( "Not Found      %x¥n" , httpdStat.NoNotFound );
    printf ( "Unknown Request %x¥n" ,
             httpdStat.NoUnknownRequest );
    printf ( "Total of transfer size %x¥n" ,
             httpdStat.NoBytes );
    }
```

IrDAを使った機器を手軽に開発するための

「IrFront H8S Trial Kit」の概要

大越章司

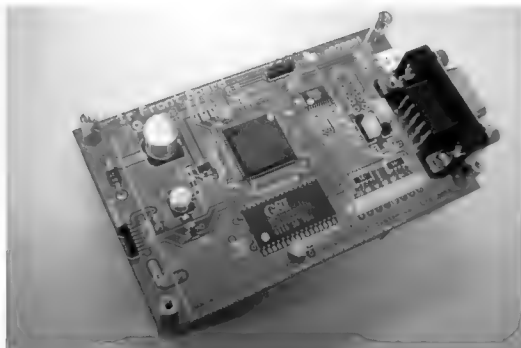
はじめに

一時はBluetoothなどの無線技術に取って代わられると見られていたIrDAだが，NTTドコモのiモード対応携帯電話機に採用されたことから，「復活」の兆しを見せている．504iシリーズで採用されたIrDAは，505iシリーズでも全機種に採用され，FOMAも含めたiアプリ対応端末の契約数は，17,195,000件に達している(NTTドコモ2003年度第1四半期オペレーションデータ2003年度実績より)．すでに自動販売機やPOSなどにIrDAを実装して決済を行うサービスが始まっているが，突如として出現した巨大マーケットをビジネスチャンスととらえる企業も多い．

開発の背景

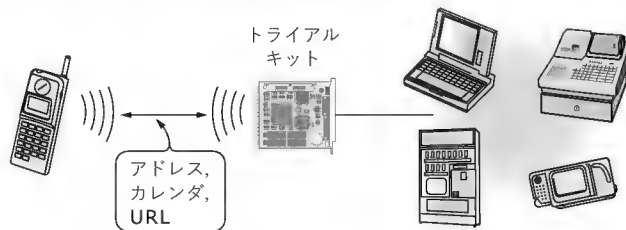
IrDAプロトコルスタック「IrFront」(開発：ACCESS)は，上述のNTTドコモのiモード対応携帯電話機にも採用され，対抗機への採用にあたっても適切な選択といえる．しかし，とりあえず新規ビジネスとして企画しようとしても，ユーザーが開発中の機器にIrDAインターフェースを組み込むのはすぐにできることでもないし，プロトコルスタックも別途用意する必要がある．そのため，スタックの実機上への移植なども考えると，気軽に試してみる，というわけにいかないのが実状だった．そこでACCESSでは，IrFrontを使った携帯電話機との通信実験が手軽に行えるソリューションを提供することにした．

トライアルキットの概要

「IrFront H8S Trial Kit(IrFront H8Sトライアルキット)」(**写真1**)は，ルネサス テクノロジ製のH8Sプロセッサ，シリアルインターフェース，IrDAトランシーバを搭載した名刺大のボードに，IrDAプロトコルスタック「IrFront」をプリインストールしたものである．ユーザーはこのキットを購入すれば，面倒なハード設計，ソフト移植などの工数をいっさいかけずに，いきなり携帯電話との通信を実験できる環境を入手できる．また，同梱のCD-ROMにiアプリのサンプルプログラムも収録したので，携帯電話を使った実際のサービスイメージにきわめて近い環境で実験できる．トライアルキットのIrFrontのもう一つの特徴は，OSレスで動作しているということである．H8Sという廉価なCPUと，OSレスで動作するIrFrontの組み合わせで，ユーザーは容易に低コストの組み込みインターフェースを開発することができる．本キットは，基本的には携帯電話とPC，あるいはユーザーが開発中の実際の機器を接続し，IrFrontでのデータ転送が確実にできることを確認するための評価キットである．ただし，本機を用いての実機開発を行うことはできない．

〔写真1〕IrFront H8S評価ボード

使用方法

本機の使用方法は，大きく分けて2種類ある．一つはPCまたは実際の機器につないで携帯電話とのデータ転送を実際に行ってみること，そしてもう一つは，携帯電話用のiアプリを開発して動作の検証を行うことである．

〔図1〕PCまたは他の組み込み機器との通信

〔図2〕iアプリを使った通信

〔図3〕IrFront H8S評価ボードのブロック図

〔表1〕IrFront H8S評価ボードの機能仕様

項　目	仕　様
CPU	H8S/2148　HD64F2148A 内蔵ROM　128Kバイト 内蔵RAM　4Kバイト
SRAM	GSI製　1Mバイト(128Kバイト×8)
システムクロック	動作周波数：7.37MHz セラミックレゾネータ 村田製作所製
ターゲットへのシリアル接続	H8S/2148内蔵チャネル1(CN3)
デバッグ用シリアル接続ボード	H8S/2148内蔵チャネル0(CN2)
IrDAシリアルトランシーバ	H8S/2148内蔵チャネル2(U3)
ボーレート	115200bps
電源	安定化電源　もしくは　単3電池3個

〔表2〕「IrFront H8S Trial Kit」のパッケージ内容

番号	項　目	内　容
1	IrFront H8S評価ボード	IrFrontボードハードウェア
2	ケーブル	Dサブ9ピンケーブル(オス－メスストレート型) 1m
3	プリインストールソフト	IrFront v2.0ベーシックパッケージ
4	CD-ROM内容	IrMC/OBEX機能制限パッケージ
		シリアル通信用IrMC/OBEX対応プログラム
		iアプリサンプルソースコード
		IrFront操作説明書
		ハードウェアマニュアル(回路図含む)

● PCまたは他の組み込み機器との通信(**図1**)

RS-232-C経由でPCまたは組み込み機器と接続し，PCのターミナルソフトなどを使って簡単な命令を送出すると，IrFrontボード経由で携帯電話上のデータ(電話帳・メール・URL)をPC側に送信できる．IrFront H8S評価ボード上にはIrFrontのサブセット版がすでにインストールされているため，ユーザーサイドでは何のソフトウェアも開発する必要がない．これにより，IrFrontが携帯電話と問題なくデータ通信できること，ユーザーが開発した機器でも問題なく使えることを確認できる．

● iアプリを使ったPCまたは他の組み込み機器との通信(**図2**)

携帯のデータをダウンロードできても，それだけでは面白くない．携帯上のiアプリとの通信ができて初めて，新たなサービスの検討ができる．IrFront H8S Trial Kitには，iアプリのサンプルソースコードが添付されているので，これを改変してユーザーごとに特化したサービスイメージでの利用をシミュレーションできる(iアプリの作成方法についてはNTTドコモのサイトに詳細な説明がある)．

利用例

携帯電話とPOS，自動販売機をIrDAで接続して決済を行うサービスはすでに開始されているが，このほかにも，電化製品のリモコンとして利用したり，カラオケのデータ入力，駐車場での料金決済，血圧計や体重計のデータを携帯に取り込んで記録・グラフ化するなど，応用範囲は無限にある．

IrDAはハードウェアコストが安価なのが魅力で，今後無線LAN技術のハードウェアコストが劇的に下がってきても，しばらくは優位を保つと考えられる．欧州の携帯電話でもすでにIrDAは採用されている(ただ，決済などの用途には使われていないようだ)し，今後日本でも利用が広がると見込める．

ハードウェア/ソフトウェアの概要とパッケージ

IrFront H8S評価ボードは，H8S/2148を採用し，ローム製IrDAモジュール，RS-232-Cインターフェース，LEDなどを55×90mmのボード上にコンパクトに実装した．外部からの3.3Vの電源で動作するとともに，ボード裏に単3電池3本用の電池ホルダを実装し，電池駆動も可能である．H8S/2148の内部メモリはROM128Kバイト/RAM4Kバイトと，IrFrontの動作環境には足りないため，外部に1MバイトのSRAMを増設している．同梱CD-ROMには回路図も添付されるため，量産への移行はスムースに行える．なお，実機への搭載を想定していないため，動作温度などの環境条件は一般的なオフィス環境となっている．IrFront H8S評価ボードのブロック図を**図3**に，機能仕様を**表1**に示す．

IrFrontは組み込み向けのコンパクトな赤外線通信プロトコルスタックで，IrMC/IrOBEX/OBEXなど豊富なオプションを用意している．今回のトライアルキットには，non-OS版のIrFront v2.0を，IrMC/OBEXの機能制限版モジュールとともに搭載している．NTTドコモの504i/505iシリーズでサポートされているiアプリ連携機能「vTrigger」に対応しており，手軽に携帯電話機との通信実験を行うことができる．「IrFront H8S Trial kit」のパッケージ内容を**表2**にまとめる．なお，次号以降で，IrDA対応システムの実験の詳細を解説する予定である．

おおこし・しょうじ　(株)ACCESS

初級ドライバ開発者のための

Windowsデバイスドライバ開発テクニック

第3回　割り込み通知の受け取りとDMA転送

丸山治雄

前回は，ドライバとアプリケーションの通信方法について解説しました．

今回は，前回の続きとして，アプリケーションがドライバから割り込み通知を受け取る簡単な方法を解説し，さらに，PLX9054のDMA機能を利用したDMA転送について解説します．

3.1 ドライバから割り込み通知を受け取る簡単な方法

ここで解説する方法は，KIT1050ボードのプッシュスイッチ割り込みをドライバが検出したら，アプリケーション側でホールドしているスレッドを再開する方法です(**リスト3.1**，**リスト3.2**)．

● 非同期動作モードでドライバをオープン

リスト3.2のSetEventSignal()内で，ドライバから通知を受け取るスレッドThreadSignal()を作成します(①)．SetEventSignal()は，スレッドの作成依頼をしてすぐに呼び出しルーチンに戻ります．

ThreadSignal()内では，まずドライバにホールドを依頼するオブジェクトhEvtを作成します(②)．このオブジェクトをDeviceIoControl()のlpOverlapped構造体に渡します(③)．

このとき，ファイルハンドルはオーバラップ(非同期動作)モードで作成する必要があります(**リスト3.3**のPCI_Open()で，OverFlag=0のときのCreateFile()を参照．①)．その理由は，たとえばハードディスクからデータを読み出すときは，アクセスが完了してからでないと次の処理に進むことができなくても問題ありませんが，割り込みのようにいつ発生するかわからないときには，アクセス依頼とアクセス完了(この場合は，割り込み発生)を切り離して処理する必要があるからです．このようなケースのときは，オーバラップモードでドライバをオープンする必要があります．

● オブジェクトのホールド処理

ドライバは，IOCTL_HOLD_REQUESTがアプリケーションからリクエストされると，KIT1050Dispatch()内のIRP_MJ_DEVICE_CONTROL処理部でホールド処理をします(①)．

まず，マルチプロセッサに対応するため，IoAcquireCancelSpinLock()で同期を取ります(②)．次に，IoSetCancelRoutine()で，ホールドしているイベントを取り消すルーチンを指定します(③．**リスト3.4**参照)．これは，アプリケーションが取り消されたときに，デバイスキューに残っているIRPを強制的に削除するためで，必ず登録しなければなりません．このキャンセルルーチンを登録しておかないと，アプリケーションが異常終了したとき，デバイスキューにIRPが残ったままになり，アプリケーションが完全に終了できません．そのため，アプリケーションを再度実行しようとすると，すでに動作していると判断され，再起動ができなくなります．

キャンセルルーチンの登録が終わったら，同期を解除してデバイスオブジェクト内のWaitQueueにリクエストされたときのIRPを保存します(④)．最後に，IRPをペンディングにして，処理を終了します(⑤)．このとき，IRPがペンディング状態であることを示すために，I/OマネージャにはSTATUS_PENDING

〔**リスト3.1**〕**メッセージの分類．リリース処理とホールド処理**〔DriverEntry.c / KIT1050Dispatch()から〕

```
// リスト2.6 (第2回掲載) のつづき

  switch (irpStack->MajorFunction)
  {
// 一般IOCTL要求
  case IRP_MJ_DEVICE_CONTROL: ←──①

    ioControlCode = irpStack
                  ->Parameters.DeviceIoControl.IoControlCode;

// KIT1050_PCI vendor Command
    if ( (ioControlCode & (USERIOCODE << 16) )
                         == (USERIOCODE << 16) )
      return( KIT1050DeviceControl( DeviceObject, Irp ) );
```

```
    switch (ioControlCode)
    {

// Thread Hold request
      case IOCTL_HOLD_REQUEST:
        IoAcquireCancelSpinLock(&kCancelSpin); ←──②
        IoSetCancelRoutine(Irp, KIT1050IntCancel); ←──③
        IoReleaseCancelSpinLock(kCancelSpin);

        // Now insert in the Irp into our queue.                  ┐
        ExInterlockedInsertTailList(&extension->WaitQueue,        │
                    &Irp->Tail.Overlay.ListEntry,                 ├④
                    &extension->QueueSpin);                       ┘

        // Mark the Irp as pending.
```

〔リスト3.1〕メッセージの分類，リリース処理とホールド処理〔DriverEntry.c / KIT1050Dispatch()から〕(つづき)

```
        IoMarkIrpPending(Irp); ←⑤

        extension->nCount = 1;
        return( STATUS_PENDING ); ←⑥

// Release Thread Hold request
      case IOCTL_RELEASE_REQUEST:
        {
          PULONG  pData;
          LONG  outputBufferLength;

          extension->nCount = 0;
          if (IsListEmpty (&extension->WaitQueue)) ←⑦
          {
            IoCompleteRequest (Irp, IO_NO_INCREMENT) ;
            return(STATUS_SUCCESS);
          }

          IoAcquireCancelSpinLock (&kCancelSpin) ;

          head = ExInterlockedRemoveHeadList
                       (&extension->WaitQueue,
                        &extension->QueueSpin) ;
          NewIrp = CONTAINING_RECORD (head, IRP,
                        Tail.Overlay.ListEntry); ←⑧

          IoSetCancelRoutine (NewIrp, NULL) ; ←⑨
          IoReleaseCancelSpinLock (kCancelSpin) ;

            irpStack = IoGetCurrentIrpStackLocation(NewIrp);
            outputBufferLength =
               irpStack
                ->Parameters.DeviceIoControl.OutputBufferLength;
            if ( outputBufferLength >= sizeof(LONG) )
            {
              pData = (PULONG)NewIrp->AssociatedIrp.SystemBuffer;
              *pData = -1;
              NewIrp->IoStatus.Information = sizeof(LONG);
            }
            else
              NewIrp->IoStatus.Information = 0;
            NewIrp->IoStatus.Status = STATUS_SUCCESS ;
            IoCompleteRequest(NewIrp, IO_NO_INCREMENT) ;
          }
          IoCompleteRequest(Irp, IO_NO_INCREMENT);
          return(STATUS_SUCCESS);

        case IOCTL_DO_NOTHING:
        {
          OVERLAPPED *Ovl = (PVOID)Irp
                            ->AssociatedIrp.SystemBuffer;
          Irp->IoStatus.Information = 0 ;
          Irp->IoStatus.Status = STATUS_SUCCESS ;
          IoCompleteRequest (Irp, IO_NO_INCREMENT) ;
          return( STATUS_PENDING );
        }
```

〔リスト3.2〕アプリケーション側の処理(NTSIGNAL.Cから)

```
typedef struct
{
  HANDLE  handle;
  PVOID  Task;
} ThreadParam, *PThreadParam;

ThreadParam  threadParam;

//extern  PKIT1050_REGISTER  PCIRegPointer;
extern  HANDLE  hPCIHdl;      // Driver handle
extern  LONG  OverFlag;       // オーバラップモード

VOID ThreadSignal( PThreadParam lpParam);
VOID EventSignalled( VOID(CALLBACK *)(ULONG), ULONG );

LONG SetEventSignal( PVOID Task )
{
  HANDLE  hThread;
  ULONG  dwThreadId;

  if ( OverFlag != 0 )  // オーバラップモードでない
    return( PCIERR_THREADCANCEL );

  threadParam.handle = hPCIHdl;
  threadParam.Task    = Task;
  hThread = CreateThread(NULL, 0,                          ┐
             (LPTHREAD_START_ROUTINE)ThreadSignal,         ├←①
             &threadParam, 0, &dwThreadId) ;               ┘
  if ( hThread == NULL )
    return( PCIERR_THREADCANCEL );
  return( PCIERR_SUCCESS );
}

VOID ThreadSignal( PThreadParam lpParam)
{
  HANDLE  hEvt;
  OVERLAPPED ovl ;
  PVOID  Task = (PVOID)lpParam->Task;
  ULONG  junk;
  ULONG  ReleaseSts = -1;


  hEvt = CreateEvent(NULL, TRUE, FALSE, NULL); ←②
  ovl.hEvent = hEvt;
  ovl.InternalHigh = 0;

  // このスレッドをHoldする。リリースされたときReleaseStsに
  // ドライバからのステータスが渡される
  // 受信バイト数は、ovl.InternalHighにセットされている
  DeviceIoControl( hPCIHdl, IOCTL_HOLD_REQUEST,  ┐
              NULL, 0,                           │
              &ReleaseSts, sizeof(LONG),         ├←③
              &junk, &ovl);                      ┘

  // スレッドをサスペンドする
  WaitForSingleObject(hEvt, INFINITE); ←④
  ResetEvent( hEvt );

  // ドライバがリリースした条件を調べる
  if ( (ovl.InternalHigh == sizeof(LONG)) &&
                                    // ドライバがリリースした
     (ReleaseSts != -1) )           // RELEASE_REQUESTではない
  {
    // 処理タスクを起動
    EventSignalled( (PVOID)Task, (ULONG)ReleaseSts );
  }
}

LONG ReleaseHoleRequest( VOID )
{
  OVERLAPPED ovl;
  HANDLE  hEvt;
  DWORD  junk;

  if ( OverFlag != 0 )  // オーバラップモードでない
    return( PCIERR_THREADCANCEL );

  hEvt = CreateEvent(NULL, TRUE, FALSE, NULL) ;
  ovl.hEvent = hEvt ;

  DeviceIoControl(hPCIHdl, IOCTL_RELEASE_REQUEST,  ┐
              NULL, 0,                             │
              NULL, 0,                             ├←⑤
              &junk, &ovl );                       ┘

  ResetEvent(hEvt);

  return( 0 );
}

VOID EventSignalled( VOID(CALLBACK *Task)(ULONG), ULONG Param )
{
  if ( Task == NULL )
    SendMessage( hWndMain, WM_COMMAND, IDM_IRQ_REQ,
                                        (LPARAM)Param );
  else
    (*Task)( Param );
}
```

〔リスト3.3〕アプリケーション側の処理（PCIFIND.Cから）

```
// PCIボードのメモリにマッピング
PKIT1050_REGISTER  PCIRegPointer = NULL;
PPCI_PLX_CONFIG    PCI9054RegAddr = NULL;
HANDLE  hPCIHdl = NULL;       // Driver handle
LONG  PCIBoardNumber = 0;     // PCI Board Number
ULONG  PCIPointer[8] =        // PCI Memory pointer
    // [0]=Memory [1]=Register [2]=Resv [3]=90X0memory
    // [4]=Port [7]=No
   { 0, 0, 0, 0, 0, 0, 0, (ULONG)-1};

ULONG  PCISize[8] =        // PCI Memory Size
    // [0]=SRAM [1]=Register [2]=Resv [5]=全Memory
    // [6]=90X0memory [7]=No
   { 0, 0, 0, 0, 0, 0, 0, 0 };

LONG  Driver_CorCPP = 0;    // Driver type 0=C 1=C++
LONG  OverFlag = 0;       // Overlap Mode 0=Overlap 1=Normal

HANDLE WINAPI PCI_Open( VOID )
{
  ULONG  cbBytesReturned;
  LONG  PciCnt = 0;
  UCHAR  string[80];

  strcpy( string, "¥¥¥¥.¥¥" );
  strcat( string, DRIVER_SERVICE_NAME );

  if ( OverFlag == 0 )
  {
    hPCIHdl = CreateFile( string,
              GENERIC_READ | GENERIC_WRITE,
              FILE_SHARE_READ | FILE_SHARE_WRITE,
              NULL,
              OPEN_EXISTING,
              FILE_FLAG_OVERLAPPED,  ←――――――――――――①
                          // 割り込み通知を受け取るときは必須
              NULL );
  }
  else
  {
    hPCIHdl = CreateFile( string,
              GENERIC_READ | GENERIC_WRITE,
              FILE_SHARE_READ | FILE_SHARE_WRITE,
              NULL,
              OPEN_EXISTING,
              FILE_ATTRIBUTE_NORMAL,
                          // 割り込み通知は受け取れない
              NULL );
  }
  if ( hPCIHdl == INVALID_HANDLE_VALUE )
  {
    hPCIHdl = (HANDLE)NULL;
    return( NULL );
  }

//  PCIボード数の取得
  DeviceIoControl(hPCIHdl, IOCTL_PCI_BOARDNUMBER,
          (LPVOID)NULL, 0,
          (LPVOID)&PCIBoardNumber, sizeof( LONG ),
          &cbBytesReturned, NULL);

  if ( PCIBoardNumber == 0 )
  {
    CloseHandle(hPCIHdl);
    hPCIHdl = NULL;
    return( NULL );
  }

// PCI Configの取得
// PCI_GetConfig( 0, &PCIConfig );

// PCIメモリアドレスの取得 */
  DeviceIoControl(hPCIHdl, IOCTL_PCI_MMAP,
           (LPVOID)&PciCnt, sizeof( LONG ),
           (LPVOID)&PCIPointer[0], sizeof(LONG) * 8,
           &cbBytesReturned, NULL);
  PCIPointer[7] = (ULONG)PciCnt;
  PCIRegPointer = (PKIT1050_REGISTER)PCIPointer[0];
  PCI9054RegAddr = (PPCI_PLX_CONFIG)PCIPointer[6];

  // メモリサイズ
  DeviceIoControl(hPCIHdl, IOCTL_PCI_MEMLENGTH,
           (LPVOID)NULL, 0,
           (LPVOID)&PCISize, sizeof(LONG) * 8,
           &cbBytesReturned, NULL);

  PciCnt++;

  return ( hPCIHdl );
}

LONG WINAPI PCI_Close( VOID )
{

  if ( hPCIHdl != 0 )
  {
    CloseHandle(hPCIHdl);
  }
  hPCIHdl = 0;
  PCIBoardNumber = 0;

  return( 0 );
}

LONG WINAPI PCI_MemorySize( PULONG MemSize )
{

  memcpy( MemSize, &PCISize, sizeof(LONG) * 8 );
  return( 0 );
}
```

を返します(⑥).

なお，WaitQueueとQueueSpinは，以下のように事前に初期化しておく必要があります．初期化はDriverEntry()内で行うのが一般的です．

InitializeListHead(&pExtension->WaitQueue);

KeInitializeSpinLock(&pExtension->QueueSpin);

アプリケーション(**リスト3.2**)は，ホールドリクエストが完了したら，WaitForSingleObject()によりイベントをウェイトさせます(④)．このウェイトは，KIT1050DpcRoutine()により，KIT1050ボードのプッシュスイッチの割り込み処理完了により解除されるか，アプリケーションにより，リリースリクエストが発行されたときに解除されます．これ以外はドライバ内部で強制的に削除されます．ドライバ内で強制的にホールドリクエストが解除されたときは，アプリケーションには通知されません．

● オブジェクトのリリース処理

登録済み(ペンディング)のIRPをアプリケーションから削除します．アプリケーションを終了するときや，割り込み処理を取り消すときに呼び出します．**リスト3.2**のReleaseHoleRequest()のように，アプリケーションからIOCTL_RELEASE_REQUESTをドライバにリクエストします(⑤)．このリクエストはすぐに戻ります．

リリースされた結果は，ThreadSignal()のWaitForSingleObject()が解除され，直前にIRPのホールドをしているDeviceIoControl()のovlパラメータにドライバからのデータサイズ(今回は4バイト)がセットされ，ReleaseSts変数に-1がドライバから渡されます．

ドライバは，IOCTL_RELEASE_REQUESTを受け取ると，次のような動作を行います．

〔リスト 3.4〕キューに残っているIRPを強制削除(PCIINTER.Cから)

```
VOID KIT1050CancelSup(IN PIRP Irp)
{
  Irp->IoStatus.Status = STATUS_CANCELLED ;
  Irp->IoStatus.Information = 0 ;

  IoCompleteRequest(Irp, IO_NO_INCREMENT);
}

VOID KIT1050IntCancel(
                    IN PDEVICE_OBJECT DeviceObject, IN PIRP Irp)
{
   KIRQL kOld ;
   PDEVICE_EXTENSION extension = DeviceObject->DeviceExtension;

   // Acquire the spin lock for the wait queue.
  KeAcquireSpinLock (&extension->QueueSpin, &kOld);

   // Now remove the irp from the queue completely.
  RemoveEntryList (&Irp->Tail.Overlay.ListEntry);

   // Release the spin lock.
  KeReleaseSpinLock (&extension->QueueSpin, kOld);

   // Release the cancel spin lock.
  IoReleaseCancelSpinLock (Irp->CancelIrql);

  KIT1050CancelSup(Irp);
}
```

1. まず，ホールドしているIRPの有無をIsListEmpty()により調べます(**リスト 3.1**の⑦).
2. ドライバがホールドしているIRPがあるときは，NewIRPにIRPを取り出し，キャンセルルーチンを削除します．ここで，キャンセルルーチンを削除しないとアプリケーションが終了したとき，I/OマネージャからKIT1050IntCancel()が呼び出されます．呼び出されたときに渡されるIRPはすでに実体がないので，誤動作の原因となります.
3. IRPスタックの中から，アプリケーションからリリース時のデータが要求されているかを調べます．もし要求されていたら，必要なデータをアプリケーションに転送する処理をします．この処理は，割り込み発生時にアプリケーションに通知する方法(**リスト 2.12**，第2回掲載)と同じですが，この場合はリリース要求に基づくリリースなので，アプリケーションには-1を転送します．アプリケーションは-1を受け取ると，何もせずにスレッドを終了させます.

3.2 PLX9054のDMA機能を利用したDMA転送

PLX9054のDMA機能を使用してKIT1050のSRAMとPCのメインメモリとの間でDMA転送を行う方法を説明します.

DMA転送を行うには，次の点に注意が必要です.

1. 転送に使用するPCのメモリは物理アドレスを使用する必要があるので，メモリアドレスを論理アドレスから物理アドレスに変換する必要があります.

   また，KIT1050のSRAMは，ローカルメモリアドレスを使用します．PCIコンフィグレーションレジスタのベースアドレスレジスタに設定されたアドレスではないことに注意してください.

   KIT1050のメモリ用のローカルアドレスは0x00000000に設定されており，さらにSRAMのうち，SRAM0は0x00200000，SRAM1は0x00300000に設定されているので，このアドレスを先頭アドレスとして使用します.
2. i386系のWindowsシステムでは，1回のDMAで転送できるデータサイズは4096バイト(ページ)以下に制限されています．また，転送開始の物理アドレスもページ単位(4Kバイトで割り切れるアドレス)にする必要があります.

「2.」の問題を解決するためにPLX9054では，ディスクリプタテーブルと呼ばれるDMA転送用の専用テーブルを作成する必要があります．また，PLX9054では2種類の転送方法があり，どちらの転送方法を使用するかによってディスクリプタテーブルの作成方法が異なります.

転送方法の一つは，1回の転送(4096バイト以下)が終了するごとに次の転送テーブルをPLX9054のレジスタに設定する方式です．もう一つの方法は，転送用のディスクリプタテーブルをすべて作成しておき，ディスクリプタテーブルの先頭アドレスをPLX9054のレジスタにセットするだけであとはPLX9054がテーブルの内容に従いすべてを制御してくれる方法です.

おのおのの方法の長所と短所は次のようになります.

毎回設定する方式は，大量のデータを転送しようとすると，テーブルの数だけ転送終了を監視しなければならないので，プログラムが煩雑になることが難点ですが，DMA動作が分割されるので他の機器への影響が少ないという利点があります.

連続転送方式は，テーブルさえ作成してしまえば，あとはPLX9054が勝手に動作するので，プログラムの負荷はかなり軽減されます．しかし，DMA動作が連続するため，他の機器の動作に悪影響を与える可能性があります[注3.1].

もちろん，PCIの仕様に準拠してレイテンシタイマレジスタによる打ち切り，あるいは256ワードの転送でバスがいったんは解放されますが，すぐにPCIバスを獲得したときは，見かけ上DMA動作が連続してしまい，とりわけ画面表示への悪影響(マウスの移動が悪くなる，ウインドウ画面の切り替えに時間がかかるなど)が確認されています.

どちらの方法を使用するかは，作成するPCIボードの目的により判断してください．連続転送のPLX9054レジスタの設定方法を**リスト 3.5**に示します．コメントになっている部分は，毎回設定方式です[注3.2].

注3.1：PLX9080(一つ前のバージョン)では，連続転送方式のディスクリプタテーブルを作成して，そのテーブルが一つしかないとき(つまり4096バイト以下の転送)は，毎回設定方式を使用すること．これは，PLX9080の不具合と考えられる．PLX9054ではこの問題は解決している.

DOSエクステンダでハードウェアデバッグ

Column

PCIボードのハードウェアデバッグには，DOSエクステンダを利用すると以下のようなメリットがあり，とても便利です．

(1) DOSなのでシステムの立ち上がりが高速．
(2) PCIコンフィグレーションレジスタの内容がわかれば，ベースアドレスの内容がPCIボードのメモリアドレスとしてそのまま使用できるため，簡単にプログラムを作成できる[注A]．
(3) 筆者は，DOS用のコンパイラとして，Watcom Cコンパイラ Ver.11.0を使用している．Watcom Cコンパイラは，同社のWebサイトから無償でダウンロードできる．
(4) Watcom Cコンパイラのデバッグ機能，または筆者作成のPCICHECKを使用すれば，プログラムを作成しなくても簡単なメモリレジスタアクセスやメモリダンプが行える．

このようにして，DOS上でハードウェアデバッグをほぼ完了させてからソフトウェア開発者に引き渡せば，デバイスドライバを含めたアプリケーションの開発もスムーズに進みます．

また，問題が発生したときに，DOSで簡単に確認作業ができるので，ハードウェアとソフトウェアのどちらに問題があるのか簡単に切り分けができます．

注A：筆者は，PCIコンフィグレーションレジスタの内容を簡単に調べるプログラムPCICHECKをWeb上で公開している．
`http://www2.ttcn.ne.jp/~Driver-YA/`

● ディスクリプタテーブルの作成

ドライバでディスクリプタテーブルを作成するには，アプリケーション側で，次の準備を行う必要があります(**リスト3.5**)．

1. 転送サイズを決める
2. 転送に必要な領域を確保する
3. 転送方向を決める

リスト3.5のサンプルでは，`DmaDescriptor`構造体のメンバに，それぞれ`PCILocalOffset`にKIT1050のSRAMアドレス，`ChainMode`に転送方向を設定し，`SrcLRB`構造体に，PCメモリアドレスと転送長を設定しています(①)．

これをもとに，ドライバでディスクリプタテーブルを作成します(**リスト3.6**．変数の説明は**表3.1**)

まず，`IoAllocateMdl()`，`MmBuildMdlForNonPagedPool()`，`MmProbeAndLockPages()`により，アプリケーションがDMA転送に使用する領域のMDL(Memory Descriptor List)を作成し，領域をロックします(①)．これで，転送用領域は，PCの物理メモリとして確保されます．したがって，この領域は，不要になったら(転送が終了)ロックの解除と解放を必ず行ってください．行わない場合，メモリリークなどを起こし，システムが誤動作する原因になります．

次に，ディスクリプタテーブル用の領域を確保します．`Pages`変数に，テーブルの数を計算して格納します(②)．テーブルの数は，

$$転送バイト数/4096 + n$$

で計算されます．nは，$0 \leqq n \leqq 2$になります．

`MmAllocateContiguousMemory()`で連続した領域を確

注3.2：DMA転送をDOSエクステンダでテストするときは，状況が変わる．DOSエクステンダにはページの概念がないので，PCのメモリを4096バイト単位に分割する必要はない．したがって変数のポインタ=物理アドレスなので，PLX9054のレジスタに物理アドレスと転送サイズを設定すれば，DMA転送を行うことができる．PCIボードのメモリはWindowsのときと同じ．

〔**リスト3.5**〕**連続転送のPLX9054レジスタの設定**(`DMACTRL.C`から)

```
// PCIボードのメモリにマッピング
extern  PKIT1050_REGISTER  PCIRegPointer;
extern  PPCI90X0_CONFIG    PCI9054RegAddr;
extern  HANDLE  hPCIHdl;        // Driver handle
extern  ULONG  PCIPointer[8];  // PCI Memory pointer
extern  LONG  OverFlag;         // Overlap Mode

ULONG WINAPI PCI_DMAMemoryRead( ULONG Board, ULONG Address,
   ULONG Size, PVOID Buff )
{
  PULONG  volatile RegAddr;
  ULONG  DmaStatus;
  DMA_LRB_REQUEST  DmaDescriptor;
  PDMA_DESCRIPTOR  DmaLink;
  LRB     SrcLRB;
  LRB     UnlockLRB;
  ULONG   Ics;

  if ( (DWORD)PCIPointer[7] == -1 )
    return ( (DWORD)PCIERR_NOBOARD );

  DmaDescriptor.PCILocalOffset = KIT1050_PHYSMEMORY + Address;  ┐
  DmaDescriptor.ChainMode = (PCI90X0_DIRECT_LOCAL_PCI |          │
               PCI90X0_DESCRIPTOR_LOCAT);                          ├①
  DmaDescriptor.Descriptor = (PLRB)&SrcLRB;                        │
  DmaDescriptor.UnlockDescriptor = (PLRB)&UnlockLRB;               │
  SrcLRB.XferBuffer = Buff;                                        │
  SrcLRB.XferLength = Size;                                        ┘
// Memoryを物理メモリに変換しMDLを作成し領域をロックする
  if ( DMA_LockBuffer( &DmaDescriptor ) == FALSE )
    return( PCIERR_DMAFAILED );
  DmaLink = SrcLRB.DescBuffer;

//
// 連続転送方式
//
  // DMAを初期化する
  // 連続転送モード
  PCI9054RegAddr->DMA0_MODE_REG = PCI90X0_INI_DMA_MODE;
  PCI9054RegAddr->DMA0_MODE_REG |= (PCI90X0_DMA_CHAINING |
                    PCI90X0_DMA_INTSELECT );
  PCI9054RegAddr->DMA0_DESCRIPTOR_REG =
                              (ULONG)SrcLRB.DescAddress;
  PCI9054RegAddr->DMA0_DESCRIPTOR_REG |=
                              DmaDescriptor.ChainMode;

  // 転送開始
  PCI9054RegAddr->DMA0_COMMAND_REG = PCI90X0_DMA_GO;
  RegAddr = (PULONG)&PCI9054RegAddr->DMA0_COMMAND_REG;
  // 転送終了を待つ
```

〔リスト3.5〕連続転送のPLX9054レジスタの設定(DMACTRL.Cから)(つづき)

```
  while( 1 )
  {
    DmaStatus = (volatile)*RegAddr;
    if( (DmaStatus & PCI90X0_DMA_DONE) != 0 )
      break;
  }

  DMA_UnlockBuffer( &UnlockLRB );
  return( Size );

//
// 毎回設定方式
//
/*
  // DMAを初期化する
  PCI9054RegAddr->DMA0_MODE_REG = PCI90X0_INI_DMA_MODE;
  // directionをSRAM->PCに設定
  PCI9054RegAddr->DMA0_DESCRIPTOR_REG =
                              PCI90X0_DIRECT_LOCAL_PCI;

  // DMA 転送開始
  while( 1 )
  {
  // DMA転送アドレス、サイズをセットする
    PCI9054RegAddr->DMA0_PCI_ADDR_REG = DmaLink->PCIAddress;
    PCI9054RegAddr->DMA0_LOCAL_ADDR_REG = DmaLink->LocalAddress;
    PCI9054RegAddr->DMA0_TRNS_COUNT_REG = DmaLink->Length;

  // 転送開始
    PCI9054RegAddr->DMA0_COMMAND_REG = PCI90X0_DMA_GO;
    RegAddr = (PULONG)((ULONG)PCI9054RegAddr +
               PCI90X0_DMA0_COMMAND_REG);

  // 転送終了を待つ
    while( 1 )
    {
      DmaStatus = (volatile)*RegAddr;
      if( (DmaStatus & PCI90X0_DMA_DONE) != 0 )
        break;
    }
    if ( (DmaLink->LinkAddress & PCI90XX_ENDofCHAIN) != 0 )
      break;
    DmaLink++;
  }

  DMA_UnlockBuffer( &UnlockLRB );
  return( Size );
*/

}

ULONG WINAPI PCI_DMAMemoryWrite( ULONG Board, ULONG Address,
  ULONG Size, PVOID Buff )
{
  PULONG  volatile RegAddr;
  ULONG  DmaStatus;
  DMA_LRB_REQUEST  DmaDescriptor;
  PDMA_DESCRIPTOR  DmaLink;
  LRB    SrcLRB;
  LRB    UnlockLRB;

  if ( (DWORD)PCIPointer[7] == -1 )
    return ( (DWORD)PCIERR_NOBOARD );

  DmaDescriptor.PCILocalOffset = KIT1050_PHYSMEMORY + Address;
  DmaDescriptor.ChainMode = (PCI90X0_DIRECT_PCI_LOCAL |
               PCI90X0_DESCRIPTOR_LOCAT);
  DmaDescriptor.Descriptor = (PLRB)&SrcLRB;
  DmaDescriptor.UnlockDescriptor = (PLRB)&UnlockLRB;
  SrcLRB.XferBuffer = Buff;
  SrcLRB.XferLength = Size;
// Memoryを物理メモリに変換しMDLを作成し領域をロックする
  if ( DMA_LockBuffer( &DmaDescriptor ) == FALSE )
    return( PCIERR_DMAFAILED );
  DmaLink = SrcLRB.DescBuffer;

//
// 連続転送方式
//
  // DMAを初期化する
  // 連続転送モード
  PCI9054RegAddr->DMA0_MODE_REG = PCI90X0_INI_DMA_MODE;
  PCI9054RegAddr->DMA0_MODE_REG |= (PCI90X0_DMA_CHAINING |
                  PCI90X0_DMA_INTSELECT );
  PCI9054RegAddr->DMA0_DESCRIPTOR_REG =
                                   (ULONG)SrcLRB.DescAddress;
  PCI9054RegAddr->DMA0_DESCRIPTOR_REG |=
                                    DmaDescriptor.ChainMode;

  // 転送開始
  PCI9054RegAddr->DMA0_COMMAND_REG = PCI90X0_DMA_GO;

  // 転送終了を待つ
  RegAddr = (PULONG)&PCI9054RegAddr->DMA0_COMMAND_REG;
  while( 1 )
  {
    DmaStatus = (volatile)*RegAddr;
    if( (DmaStatus & PCI90X0_DMA_DONE) != 0 )
      break;
  }

  DMA_UnlockBuffer( &UnlockLRB );
  return( Size );

//
// 毎回設定方式
//
/*
  // DMAを初期化する
  PCI9054RegAddr->DMA0_MODE_REG = PCI90X0_INI_DMA_MODE;
  // directionをPC->SRAMに設定
  PCI9054RegAddr->DMA0_DESCRIPTOR_REG =
                                   PCI90X0_DIRECT_PCI_LOCAL;

  // DMA 転送開始
  while( 1 )
  {
  // DMA転送アドレス、サイズをセットする
    PCI9054RegAddr->DMA0_PCI_ADDR_REG = DmaLink->PCIAddress;
    PCI9054RegAddr->DMA0_LOCAL_ADDR_REG = DmaLink->LocalAddress;
    PCI9054RegAddr->DMA0_TRNS_COUNT_REG = DmaLink->Length;

  // 転送開始
    PCI9054RegAddr->DMA0_COMMAND_REG = PCI90X0_DMA_GO;
    RegAddr = (PULONG)((ULONG)PCI9054RegAddr +
         PCI90X0_DMA0_COMMAND_REG);

  // 転送終了を待つ
    while( 1 )
    {
      DmaStatus = (volatile)*RegAddr;
      if( (DmaStatus & PCI90X0_DMA_DONE) != 0 )
        break;
    }
    if ( (DmaLink->LinkAddress & PCI90XX_ENDofCHAIN) != 0 )
      break;
    DmaLink++;
  }

  DMA_UnlockBuffer( &UnlockLRB );
  return( Size );
*/

}

BOOLEAN WINAPI DMA_LockBuffer( PDMA_LRB_REQUEST lpLRB )
{
  DWORD   dwReturned ;

  return( DeviceIoControl( hPCIHdl, IOCTL_LOCK,
            lpLRB, sizeof(DMA_LRB_REQUEST),
            lpLRB, sizeof(DMA_LRB_REQUEST),
            &dwReturned, NULL ) );
}

BOOLEAN WINAPI DMA_UnlockBuffer( PLRB lpLRB  )
{
  DWORD   dwReturned ;
  return( DeviceIoControl( hPCIHdl, IOCTL_UNLOCK,
            lpLRB, sizeof(LRB),
            NULL, 0,
            &dwReturned, NULL ) );
}
```

〔リスト3.6〕ディスクリプタテーブルの作成と解除〔DriverEntry.c / KIT1050Dispatch()から〕

```
// リスト3.1の続き

// DMA用ディスクリプタテーブルの作成
      case IOCTL_LOCK:
      {
        PDMA_LRB_REQUEST  Lrb_Req = (PDMA_LRB_REQUEST)IoBuffer;
        PLRB         lrb;
        PLRB         lrbUnlock;
        PHYSICAL_ADDRESS  Highest;
        PHYSICAL_ADDRESS  PhysicalAddress;
        PDMA_DESCRIPTOR   Descriptor;
        PUCHAR        Buffer;
        ULONG         PCIAddress;
        ULONG         ChainMode;
        ULONG          LinkAddress;
        ULONG          Length;
        ULONG          Offset;
        LONG         Pages;

        Irp->IoStatus.Information = 0;

      // 入力パラメータチェック
        if( ( InputSize  < sizeof(PDMA_LRB_REQUEST) ) ||
          ( OutputSize < sizeof(PDMA_LRB_REQUEST) ) )
        {
          Status = STATUS_INVALID_PARAMETER ;
          break ;
        }

        PCIAddress = Lrb_Req->PCILocalOffset;
        lrb = Lrb_Req->Descriptor;
        ChainMode = Lrb_Req->ChainMode;
        lrbUnlock = Lrb_Req->UnlockDescriptor;  // UNLOCK用LRB
        *lrbUnlock = *lrb;

      // 転送用バッファのMDLを作成して領域をロックする
        lrb->XferMdl = IoAllocateMdl( lrb->XferBuffer,                 ┐
                                      lrb->XferLength,                 │
                   FALSE, FALSE, NULL ) ;                              │
        if( lrb->XferMdl == NULL )                                     │
        {                                                              │
          Status = STATUS_ACCESS_DENIED;                               │
          break;                                                       ├①
        }                                                              │
        lrbUnlock->XferMdl = lrb->XferMdl;                             │
        MmBuildMdlForNonPagedPool( lrb->XferMdl ) ;                    │
        MmProbeAndLockPages( lrb->XferMdl, KernelMode,                 │
                                      IoModifyAccess);                 ┘

      // ディスクリプタ用バッファを割り当てる
        Highest.LowPart = Highest.HighPart = 0xFFFFFFFF ;
        Pages = ADDRESS_AND_SIZE_TO_SPAN_PAGES(lrb->XferBuffer,  }②
                          lrb->XferLength );
        lrb->DescLength = Pages * sizeof(DMA_DESCRIPTOR) ;
        lrbUnlock->DescLength = lrb->DescLength;
        Offset = (ULONG)MmAllocateContiguousMemory(              }③
                (lrb->DescLength + 32),  // Make Quad word
                Highest );
        if( Offset == 0 )
        {
          MmUnlockPages( lrb->XferMdl ) ;
          IoFreeMdl( lrb->XferMdl ) ;
          Status = STATUS_INVALID_LOCK_SEQUENCE;
          break ;
        }
        lrbUnlock->DescMemory = (PVOID)Offset;
        // Make Quad word address
        lrb->DescMemory = (PVOID)((Offset + 15) & 0xFFFFFFF0);

      // ディスクリプタ用バッファのMDLを作成して領域をロックする
        lrb->DescMdl = IoAllocateMdl( lrb->DescMemory,           ┐
                                      lrb->DescLength,           ├←④
                     FALSE, FALSE, NULL ) ;                      ┘
        if( lrb->DescMdl == NULL )
        {
          MmUnlockPages( lrb->XferMdl ) ;
          IoFreeMdl( lrb->XferMdl ) ;
          lrb->XferMdl = NULL;
          MmFreeContiguousMemory( lrbUnlock->DescMemory ) ;
          Status = STATUS_INVALID_LOCK_SEQUENCE;
          break ;
        }
        lrbUnlock->DescMdl = lrb->DescMdl;
        MmBuildMdlForNonPagedPool( lrb->DescMdl ) ;
        MmProbeAndLockPages( lrb->DescMdl, KernelMode,
                                      IoModifyAccess);

      // ディスクリプタの物理メモリを返す
        PhysicalAddress = MmGetPhysicalAddress(
                                      lrb->DescMemory ) ;
        lrb->DescAddress = PhysicalAddress.LowPart ;
        lrbUnlock->DescAddress = lrb->DescAddress;

      // ユーザからアクセスできるポインタに変換する
        lrb->DescBuffer = (PVOID)MapMemMapTheMemory (             ┐
                      &PhysicalAddress,                           ├←⑤
                      lrb->DescLength                             │
                      );                                          ┘
/*
        if( lrb->DescBuffer == (PVOID)-1 )
        {
          MmUnlockPages( lrb->XferMdl ) ;
          IoFreeMdl( lrb->XferMdl ) ;
          MmUnlockPages( lrb->DescMdl ) ;
          IoFreeMdl( lrb->DescMdl ) ;
          MmFreeContiguousMemory( lrbUnlock->DescMemory ) ;
          Status = STATUS_INVALID_LOCK_SEQUENCE;
          break ;
        }
        lrbUnlock->DescBuffer = lrb->DescBuffer;
*/

      // ディスクリプタリストを生成する
        Descriptor = (PDMA_DESCRIPTOR)lrb->DescMemory ;
        LinkAddress = lrb->DescAddress ;
        Buffer = lrb->XferBuffer;
        Length = lrb->XferLength;
        Offset = MmGetMdlByteOffset( lrb->XferMdl ) ;
        while( --Pages >= 0 )
        {
          Descriptor->PCIAddress =                                    }⑥
                 MmGetPhysicalAddress(Buffer).LowPart;
          Descriptor->LocalAddress = PCIAddress; ←⑦
          Descriptor->Length = ( Length < (PAGE_SIZE-Offset) )        }⑧
                       ? Length : (PAGE_SIZE-Offset) ;
          Descriptor->LinkAddress = LinkAddress +                     }←⑨
                          sizeof(DMA_DESCRIPTOR) ;
          Descriptor->LinkAddress |= ChainMode;
          PCIAddress += Descriptor->Length;
          if( Pages == 0 )
          {
            Descriptor->LinkAddress |= PCI90XX_ENDofCHAIN;
            break ;
          }
          Length -= PAGE_SIZE - Offset;
          Buffer += PAGE_SIZE - Offset;
          LinkAddress += sizeof(DMA_DESCRIPTOR);
          Descriptor ++ ;
          Offset = 0 ;
        }

      // 出力バッファへ返すサイズを設定する
        Irp->IoStatus.Information = sizeof(PDMA_LRB_REQUEST);
        Status = STATUS_SUCCESS ;
      }

      Irp->IoStatus.Status = Status;
      IoCompleteRequest (Irp, IO_NO_INCREMENT) ;
      return( Status );

// DMA用ディスクリプタテーブルの解除
      case IOCTL_UNLOCK:
      {

        PLRB  lrb = (PLRB)IoBuffer ;

        Irp->IoStatus.Information = 0 ;

      // 入力パラメータチェック
        if( InputSize < sizeof(LRB) )
        {
          Status = STATUS_INVALID_PARAMETER ;
          break ;
        }
```

〔リスト3.6〕ディスクリプタテーブルの作成と解除〔DriverEntry.c / KIT1050Dispatch()から〕（つづき）

```
    // 転送用バッファのロックを解除してMDLを解放する
      MmUnlockPages( lrb->XferMdl ) ;
      IoFreeMdl( lrb->XferMdl ) ;

    // ディスクリプタ用バッファのロックを解除してMDLを解放する
      MmUnlockPages( lrb->DescMdl ) ;  }
      IoFreeMdl( lrb->DescMdl ) ;      } ←⑩

    // アプリケーション用ディスクリプタアドレスを解放する
      if( lrb->DescBuffer != (PVOID)-1 )
      {
        ZwUnmapViewOfSection( (HANDLE)-1,
                              (PULONG)lrb->DescBuffer ); ⑪
      }
    // ディスクリプタ用MDLを解放する
      MmFreeContiguousMemory( lrb->DescMemory ) ;
      Status = STATUS_SUCCESS ;
    }
        Irp->IoStatus.Status = Status;
        IoCompleteRequest (Irp, IO_NO_INCREMENT) ;
        return( Status );
        break;

        default:
          break;
      }

    break;
  }

  Irp->IoStatus.Status = STATUS_NOT_IMPLEMENTED ;
  Irp->IoStatus.Information = 0 ;
  IoCompleteRequest(Irp, IO_NO_INCREMENT);

  return(STATUS_NOT_IMPLEMENTED) ;
}
```

〔表3.1〕変数の説明

ULONG PCIAddress	PCISRAMメモリアドレスで次の範囲になる． SRAM0では，0x00200000～0x002fffffff SRAM1では，0x00300000～0x003ffffff
ULONG LinkAddress	次のディスクリプタテーブルアドレス（物理アドレス）を表す
PDMA_DESCRIPTOR Descriptor	現在のディスクリプタテーブルのポインタ
PLRB lrb	物理メモリをロックしたときの情報で，各メンバの内容は**表3.1(b)**，**表3.1(c)**．**表3.1(c)**はシステムが使用
PLRB lrbUnlock	ディスクリプタテーブルを解放するときに使用する．内容はlrbと同じ
ULONG ChainMode	転送方向を表す

(a)

PVOID XferBuffer	転送用バッファのポインタ
ULONG XferLength	転送用バッファのバイト数
PVOID DescBuffer	ディスクリプタ用バッファのポインタ．この領域にディスクリプタテーブルが作成される
ULONG DescLength	ディスクリプタ用バッファのバイト数

(b)

LONG DescAddress	ディスクリプタ用バッファの物理アドレス
PVOID DescMemory	転送用バッファ領域
PMDL XferMdl	転送用バッファのMDL
PMDL DescMdl	ディスクリプタ用バッファのMDL

(c)

保しているのはPLX9054の仕様で，連続転送方式のときにはテーブル単位に連続したメモリを確保する必要があるためです（③）．毎回設定方式のときは，とくに連続したメモリを確保する必要はないので，他のメモリアロケーションを使用することも可能です．ディスクリプタテーブル用に確保した領域もPCの物理メモリとして確保するために，領域をIoAllocateMdl()でロックします（④）．この領域も，不要になったら必ずロックの解除と解放をしてください．

次に行っているMapMemMapTheMemory()は，アプリケーションで毎回設定方式を使用してDMA転送を行うときにディスクリプタテーブルの内容を読み出すためで，連続転送方式を使用するときやドライバ内でDMA転送を行うとき，Windows XPのときは，この部分は不要です．

最後に，確保したディスクリプタテーブル領域にテーブルを作成します．Descriptor->PCIAddressには，PCメモリの物理アドレスが格納されます（⑥）．Descriptor->LocalAddressには，KIT1050のSRAMアドレス（ローカルアドレス）が格納されます（⑦）．Descriptor->Lengthには，転送サイズが格納されます（⑧）．PCIAddressとLocalAddressは，次のアドレスを示すためにこの値が加算されます．Descriptor->LinkAddressは，連続転送方式のときに，PLX9054が使用する次のディスクリプタアドレスを表します（⑨）．LinkAddressにChainModeを付加しているのは，転送方向を付加するためです．

テーブルの最後には，LinkAddressにPCI90XX_ENDofCHAINを付加していますが，これは連続転送方式のときに，PLX9054のディスクリプタテーブル終了を表すフラグで，転送終了に使用しています．

● ディスクリプタテーブルの解除

DMA転送が終了したら，転送に使用したPCメモリ領域とディスクリプタテーブル領域のロック解除と，領域の解放を必ず行ってください．

PCメモリ領域とディスクリプタテーブル領域のMDLの解放は，リストにあるようにMmUnlockPages()とIoFreeMdl()を使用して行います（⑩）．ZwUnmapViewOfSection()を使用してDescBufferを解放しているのは毎回設定方式で，かつアプリケーションでDMA転送を行うときに使用するもので，ロックを行うときにこの領域を作成したときのみ解放してください（⑪）．Windows XPでは，この領域ポインタは-1になるので解放の必要はありません．

まるやま・はるお　ドライバ屋

やり直しのための信号数学

DCTとマルチレート信号処理

三谷政昭

前回(2003年10月号)は，DCTとフィルタバンク(複数のフィルタを並列構成したもの)の関係，フィルタバンクの物理的な意味付けを示し，さらに一般式を示してDCTによる信号解析の考え方や特徴を中心に，具体的数値例に基づき解説した．

今回は，DCT/IDCTによるディジタル信号解析を「信号を周波数帯域ごとに分割し，何らかの処理を施した後で再合成するシステム」として実現する手法を示す．本手法は，サブバンド，マルチレートとよばれる信号処理技術に基礎を置くもので，信号のサンプリング周波数を可変(間引き，補間)することによって，効率的なDCT，IDCT処理を実現するものである．なお，フィルタバンクの考え方には，ウェーブレット変換に結びつく重要な内容が含まれているので，しっかりと読み進めていってもらいたい．(筆者)

DCT，IDCTのフィルタバンク構成システム

DCT値，IDCT値を算出するための，フィルタバンクを利用した信号処理システムの一般的な構成を**図19.1**に示す．**図19.1**に示す信号処理システムは，

① 分析(アナリシスバンク：Analysis bank)

② 合成(シンセシスバンク：Synthesis bank)

の二つのフィルタバンクから構成されており，分析フィルタバンクの出力はサブバンド信号とよばれ，周波数帯域ごとに分割されている．DCT値は，このサブバンド信号に相当し，データ圧縮，適応信号処理などのさまざまな信号処理応用分野で利用されている．

まずは，サンプル数$N=2$に対するDCT，IDCT計算処理をフィルタバンク構成するときの分析と合成の部分だけを取り出して，**図2**に示す．この構成は，2分割フィルタバンクの構成で，並列構成とよばれ，フィルタバンクのもっともシンプルなものである．

それでは，2サンプルを一つのブロックとしてDCT，IDCT計算する場合を例に採り上げて，信号成分を周波数帯域ごとに分割し，得られた信号から元の信号を再合成する処理を体験してみることにしたい．

いま，ディジタル信号を，

$$x_{-1},\ x_0,\ x_1,\ x_2,\ x_3,\ x_4,\ \cdots$$

と左から右へと順に入力して得られる二つの周波数成分ごとに分割された信号を$e_m{}^{(L)}$，$e_m{}^{(H)}$とし，さらに周波数分割された信号$e_m{}^{(L)}$と$e_m{}^{(H)}$から再合成されて出力される信号を$y_{2m}\left(=y_0{}^{(m)}\right)$，$y_{2m+1}\left(=y_1{}^{(m)}\right)$と表し，次のようにブロック計算することを考えてみよう(**図19.2**)．

(i) $m=0$

$$e_0{}^{(L)}=\frac{x_0+x_{-1}}{2}$$

$$e_0{}^{(H)}=\frac{-x_0+x_{-1}}{2}$$

$$y_0=y_0{}^{(0)}=e_0{}^{(L)}+e_0{}^{(H)}=\frac{2x_{-1}}{2}=x_{-1}$$

$$y_1=y_1{}^{(0)}=e_0{}^{(L)}-e_0{}^{(H)}=\frac{2x_0}{2}=x_0$$

(ii) $m=1$

$$e_1{}^{(L)}=\frac{x_2+x_1}{2}$$

$$e_1{}^{(H)}=\frac{-x_2+x_1}{2}$$

$$y_2=y_0{}^{(1)}=e_1{}^{(L)}+e_1{}^{(H)}=\frac{2x_1}{2}=x_1$$

$$y_3=y_1{}^{(1)}=e_1{}^{(L)}-e_1{}^{(H)}=\frac{2x_2}{2}=x_2$$

(iii) $m=2$

$$e_2{}^{(L)}=\frac{x_4+x_3}{2}$$

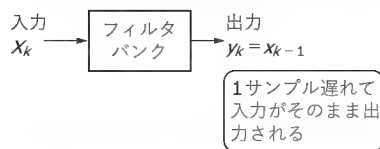
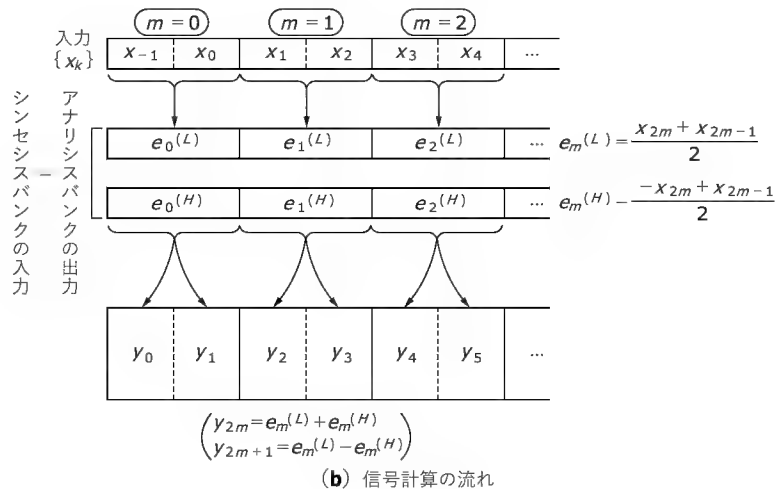
$$e_2{}^{(H)}=\frac{-x_4+x_3}{2}$$

〔図19.1〕フィルタバンク構成システムの概略図

〔図19.3〕完全再構成フィルタバンクにおける入出力関係

完全再構成条件
一般的には，n_0サンプルが遅れて入力がそのまま出力されること
$y_k = x_{k-n_0}$

〔図19.2〕DCT，IDCTのフィルタバンクによる実現

(a) システム構成

(b) 信号計算の流れ

$$y_4 = y_0^{(2)} = e_2^{(L)} + e_2^{(H)} = \frac{2x_3}{2} = x_3$$

$$y_5 = y_1^{(2)} = e_2^{(L)} - e_2^{(H)} = \frac{2x_4}{2} = x_4$$

以上の信号計算の流れを，m番目のブロックに対する一般式として示せば，次のように表される．

$$e_m^{(L)} = \frac{x_{2m} + x_{2m-1}}{2} \quad \cdots\cdots(1)$$

$$e_m^{(H)} = \frac{-x_{2m} + x_{2m-1}}{2} \quad \cdots\cdots(2)$$

$$y_{2m} = y_0^{(m)} = e_m^{(L)} + e_m^{(H)} = \frac{2x_{2m-1}}{2} = x_{2m-1} \quad \cdots\cdots(3)$$

$$y_{2m+1} = y_1^{(m)} = e_m^{(L)} - e_m^{(H)} = \frac{2x_{2m}}{2} = x_{2m} \quad \cdots\cdots(4)$$

ここで，式(1)，式(2)は“分析フィルタバンク”における信号処理で周波数成分ごとに分割する処理，式(3)，式(4)は“合成フィルタバンク”における信号処理で再合成する処理に相当する．このように，入力信号と出力信号をそれぞれ二つずつのサンプルごとに一つのブロックにまとめて計算する手法こそが，DCT，IDCT計算の効率的な処理を可能にするのである．

また，式(1)～式(4)より，入力信号のブロック$\{x_{2m-1}, x_{2m}\}$と出力信号のブロック$\{y_{2m}, y_{2m+1}\}$の間には，

$$y_{2m} = x_{2m-1} \quad \cdots\cdots(5)$$

$$y_{2m+1} = x_{(2m+1)-1} = x_{2m} \quad \cdots\cdots(6)$$

となる関係が成立し，$k = 2m, 2m+1$として，

$$y_k = x_{k-1} \quad \cdots\cdots(7)$$

とまとめて表現できる．つまり，入力信号系列が1サンプル時間だけ遅れて出力されることになり，フィルタバンク構成の妥当性が理解される(**図19.3**)．式(7)のように入力信号が単にずれてそのまま出力されるシステムは，“完全再構成フィルタバンク”といい，式(7)は“完全再構成条件”とよばれるものである．なお，一般的には，適当な正数n_0として，入力信号がn_0サンプル時間の遅れでそのまま出力される関係，すなわち，

$$y_k = x_{k-n_0}$$

が成立すればよい．

例題1

いま，入力信号系列$\{x_k\}$に対して，

$$e_m^{(L)} = x_{2m} + x_{2m-1} \quad \cdots\cdots(8)$$

$$e_m^{(H)} = -x_{2m} + x_{2m-1} \quad \cdots\cdots(9)$$

と計算して得られる二つの出力信号系列$\{e_m^{(L)}, e_m^{(H)}\}$に何らかの計算処理を施すことにより，式(7)の出力を得たい．計算処

理する式を求めよ．

解答1

まず，求めたい計算処理する式を，

$$y_{2m}=Ae_m{}^{(L)}+Be_m{}^{(H)} \quad \cdots\cdots(10)$$

$$y_{2m+1}=Ce_m{}^{(L)}+De_m{}^{(H)} \quad \cdots\cdots(11)$$

と表し，式(8)，(9)を代入整理して，式(7)の“完全再構成条件”を与える．すなわち，

$$y_{2m}=A\left(x_{2m}+x_{2m-1}\right)+B\left(-x_{2m}+x_{2m-1}\right)$$
$$=(A-B)x_{2m}+(A+B)x_{2m-1} \quad \cdots\cdots(12)$$

$$y_{2m+1}=C\left(x_{2m}+x_{2m-1}\right)+D\left(-x_{2m}+x_{2m-1}\right)$$
$$=(C-D)x_{2m}+(C+D)x_{2m-1} \quad \cdots\cdots(13)$$

の関係から，式(7)より，$k=2m$，$2m+1$として，

$$y_{2m}=x_{2m-1} \quad \cdots\cdots(14)$$

$$y_{2m+1}=x_{(2m+1)-1}=x_{2m} \quad \cdots\cdots(15)$$

となるように(A，B，C，D)を求めることに帰着される．

よって，式(12)と式(14)の係数を比較して，

$A-B=0$，$A+B=1$

となる関係から，

$A=0.5$，$B=0.5$

が得られる．同様に，式(13)と式(15)より，

$C-D=1$，$C+D=0$

となる関係から，

$C=0.5$，$D=-0.5$

となる．よって，計算処理する式は次のように求められる．

$$y_{2m}=0.5e_m{}^{(L)}+0.5e_m{}^{(H)} \quad \cdots\cdots(16)$$

$$y_{2m+1}=0.5e_m{}^{(L)}-0.5e_m{}^{(H)} \quad \cdots\cdots(17)$$

分析フィルタバンクとダウンサンプリング

ここでは，式(1)，式(2)が表す信号処理に関して，伝達関数，周波数特性，システム構成を簡単に示しておこう(2003年10月号，第18回「DCTとフィルタバンク」参照)．

手始めに式(1)，式(2)をz変換して，伝達関数はそれぞれ，

$$H^{(L)}(z)=\frac{1+z^{-1}}{2} \quad \cdots\cdots(18)$$

$$H^{(H)}(z)=\frac{-1+z^{-1}}{2} \quad \cdots\cdots(19)$$

と求められ，**図4**のようにフィルタバンクを構成できる．ここで，各フィルタの出力$\left\{e_m{}^{(L)},\ e_m{}^{(H)}\right\}$はDCT値そのものであり，

ローパスフィルタの出力 $\left(e_m{}^{(L)}\right)=C_0{}^{(2)}=\frac{1}{2}x_{2m}+\frac{1}{2}x_{2m-1}$ $\cdots\cdots(20)$

ハイパスフィルタの出力 $\left(e_m{}^{(H)}\right)=C_1{}^{(2)}=-\frac{1}{2}x_{2m}+\frac{1}{2}x_{2m-1}$ $\cdots\cdots(21)$

となる．

また，**図19.4**の二つのディジタルフィルタの振幅特性は，式(18)，式(19)の伝達関数において$z=\mathrm{e}^{j2\pi fT}$(T[秒]はサンプリング間隔)を代入して絶対値を計算すればよく，

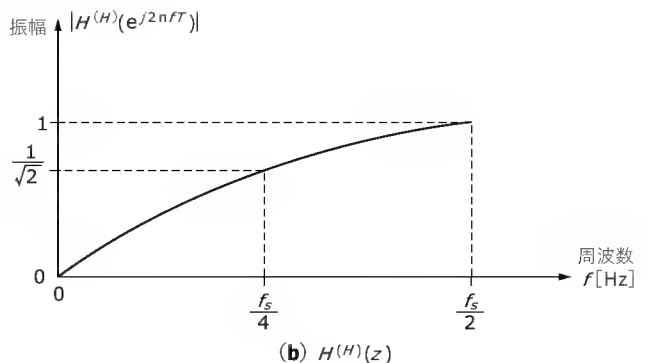
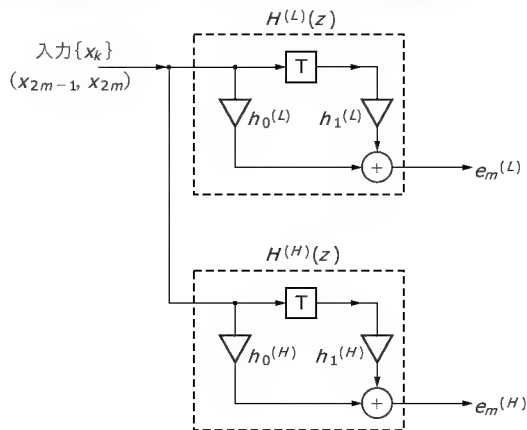
$$\left|H^{(L)}\left(\mathrm{e}^{j2\pi fT}\right)\right|=\left|\cos(\pi fT)\right| \quad \cdots\cdots(22)$$

〔図19.4〕分析フィルタバンクのシステム構成例

〔図19.5〕分析フィルタバンクの振幅特性(f_sはサンプリング周波数，$f_s=\frac{1}{T}$[Hz])

(a) $H^{(L)}(z)$

(b) $H^{(H)}(z)$

〔図 19.6〕分析フィルタバンクのシステム構成例

〔図 19.7〕合成フィルタバンクのシステム構成例

$$\left|H^{(H)}\left(e^{j2\pi fT}\right)\right|=\left|\sin(\pi fT)\right| \quad \cdots\cdots(23)$$

となる(**図 19.5**)．つまり，**図 19.5**(**a**)より，振幅特性$\left|H^{(L)}\left(e^{j2\pi fT}\right)\right|$は低い周波数成分が通りやすい“ローパス(Low Pass)フィルタ”，他方，振幅特性$\left|H^{(H)}\left(e^{j2\pi fT}\right)\right|$は**図 19.5**(**b**)より高い周波数成分が通りやすい“ハイパス(High Pass)フィルタ”であることがわかる．しかるに，ここまでの式表現における上付き文字の(L)はローパスフィルタ，(H)はハイパスフィルタを意味していることもおわかりいただけるであろう．

したがって，**図 19.2**(**a**)におけるシステム1は入力信号から低い周波数成分を抽出するためのローパスフィルタ，システム2は高い周波数成分を抽出するためのハイパスフィルタであることが導き出せる．

以上より，**図 19.4**の各フィルタは周波数帯域ごとのスペクトル成分を抽出する働きを有することになる．この分析フィルタバンクの出力(DCT 値に相当)はサブバンド信号とよばれ，応用によってさまざまな処理が施される．ここで，$H^{(H)}(z)$は，奇数番目の DCT 値を計算するフィルタなので，タップ係数は正規直交基底ベクトルの各要素の伝達関数に，(−1)を掛けたものになる．すなわち，

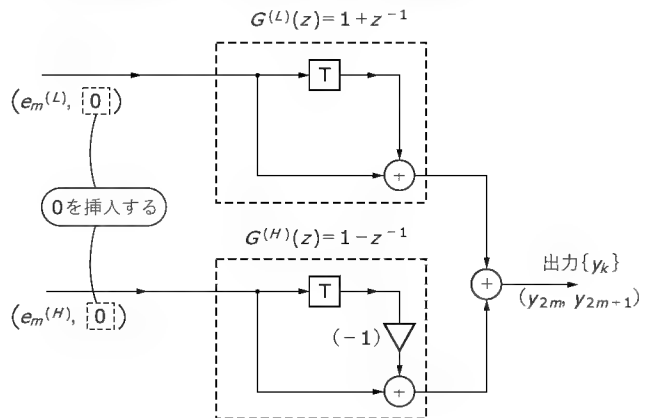
$$H^{(H)}(z)=(-1)\times\frac{1-z^{-1}}{2}=\frac{-1+z^{-1}}{2} \quad \cdots\cdots(24)$$

であり，タップ係数は正規直交基底ベクトルの各要素の正負が反転して，式(19)に一致することが確かめられる．

次に，**図 19.4**の分析フィルタバンクのローパスフィルタ，ハイパスフィルタのタップ係数をそれぞれ$\left\{h_0^{(L)},\ h_1^{(L)}\right\}$，$\left\{h_0^{(H)},\ h_1^{(H)}\right\}$とすると，伝達関数は，

$$H^{(L)}(z)=h_0^{(L)}+h_1^{(L)}z^{-1} \quad \cdots\cdots(25)$$

$$H^{(H)}(z)=h_0^{(H)}+h_1^{(H)}z^{-1} \quad \cdots\cdots(26)$$

と表されることになる(**図 19.6**)．

また，**図 19.6**の分析フィルタバンクの信号処理は，式(1)，式(2)を考慮して，

$$e_m^{(L)}=h_0^{(L)}x_{2m}+h_1^{(L)}x_{2m-1} \quad \cdots\cdots(27)$$

$$e_m^{(H)}=h_0^{(H)}x_{2m}+h_1^{(H)}x_{2m-1} \quad \cdots\cdots(28)$$

と表され，式(20)と式(21)と対比させてみることにより，

$$h_0^{(L)}=\frac{1}{2},\ h_1^{(L)}=\frac{1}{2} \quad \cdots\cdots(29)$$

$$h_0^{(H)}=-\frac{1}{2},\ h_1^{(H)}=\frac{1}{2} \quad \cdots\cdots(30)$$

で与えられる．

ところで，式(27)，式(28)の計算処理において，1ブロックの出力信号$\left\{e_m^{(L)},\ e_m^{(H)}\right\}$はそれぞれ，2個の入力信号$\left\{x_{2m-1},\ x_{2m}\right\}$から計算されるので，データ数は半分になることに気づく．言い換えれば，出力計算後のデータに対して1サンプルの間引き(デシメーション：decimation)を行う必要がある．この間引き操作を“ダウンサンプリング(down sampling)”といい，二つのフィルタによって信号を2分割したときに，全体としてデータ数を一定に保つことになる．なお，ダウンサンプリングを実現するものを“ダウンサンプラ(down sampler)”という．

合成フィルタバンクとアップサンプリング

分析フィルタバンクと対をなす合成フィルタ，すなわち式(3)，式(4)が表す信号処理に関して，伝達関数，周波数特性，システム構成を簡単に示しておこう．

まずは式(3)，式(4)をz変換して，伝達関数はそれぞれ，

$$G^{(L)}(z)=1+z^{-1} \quad \cdots\cdots(31)$$

$$G^{(H)}(z)=1-z^{-1} \quad \cdots\cdots(32)$$

と求められ，**図 19.7**のようにフィルタバンク構成できる．ここ

〔図 19.8〕合成フィルタバンクの振幅特性（f_s はサンプリング周波数，$f_s = \frac{1}{T}$[Hz]）

(a) $G^{(L)}(z)$

(b) $G^{(H)}(z)$

〔図 19.9〕合成フィルタバンクのシステム構成例

で，各フィルタの出力は IDCT 値そのものであり，

$$\text{ローパスフィルタの出力}\left(x_{2m}\right) = C_0^{(2)} + C_1^{(2)} \quad \cdots\cdots(33)$$

$$\text{ハイパスフィルタの出力}\left(x_{2m+1}\right) = C_0^{(2)} - C_1^{(2)} \quad \cdots\cdots(34)$$

となる．

また，**図 19.7** の二つのディジタルフィルタの振幅特性は，式(31)，式(32)の伝達関数において $z = e^{j2\pi fT}$ を代入して絶対値を計算すればよく，

$$\left|G^{(L)}\left(e^{j2\pi fT}\right)\right| = 2\left|\cos(\pi fT)\right| \quad \cdots\cdots(35)$$

$$\left|G^{(H)}\left(e^{j2\pi fT}\right)\right| = 2\left|\sin(\pi fT)\right| \quad \cdots\cdots(36)$$

となる（**図 19.8**）．つまり，**図 19.8(a)** より，振幅特性 $\left|H^{(L)}\left(e^{j2\pi fT}\right)\right|$ は低い周波数成分が通りやすい“ローパスフィルタ”，他方，振幅特性 $\left|H^{(H)}\left(e^{j2\pi fT}\right)\right|$ は**図 19.8(b)** より高い周波数成分が通りやすい“ハイパスフィルタ”なのである．したがって，**図 19.2(a)** におけるシステム 3 はローパスフィルタ，システム 4 はハイパスフィルタに相当する．

図 19.7 の合成フィルタバンクは，周波数帯域ごとの分割されたサブバンド信号から再合成した出力（IDCT 値に相当）を得る処理に相当する．

次に，**図 19.7** の合成フィルタバンクのローパスフィルタ，ハイパスフィルタのタップ係数をそれぞれ $\left\{g_0^{(L)}, g_1^{(L)}\right\}$，$\left\{g_0^{(H)}, g_1^{(H)}\right\}$ とすると，伝達関数は，

$$G^{(L)}(z) = g_0^{(L)} + g_1^{(L)} z^{-1} \quad \cdots\cdots(37)$$

$$G^{(H)}(z) = g_0^{(H)} + g_1^{(H)} z^{-1} \quad \cdots\cdots(38)$$

と表されることになる（**図 19.9**）．

また，**図 19.8** の合成フィルタバンクの信号処理は，式(3)，式(4)を考慮して，

$$y_{2m} = y_0^{(m)} = g_0^{(L)} e_m^{(L)} + g_0^{(H)} e_m^{(H)} \quad \cdots\cdots(39)$$

$$y_{2m+1} = y_1^{(m)} = g_1^{(L)} e_m^{(L)} + g_1^{(H)} e_m^{(H)} \quad \cdots\cdots(40)$$

と表され，式(20)，式(21)より得られる関係，すなわち，

$$e_m^{(L)} = C_0^{(2)},\ e_m^{(H)} = C_1^{(2)} \quad \cdots\cdots(41)$$

を考慮すれば，完全再構成条件〔式(5)，式(6)〕より，

$$y_{2m} = g_0^{(L)} C_0^{(2)} + g_0^{(H)} C_1^{(2)} = x_{2m-1} \quad \cdots\cdots(42)$$

$$y_{2m+1} = g_1^{(L)} C_0^{(2)} + g_1^{(H)} C_1^{(2)} = x_{2m} \quad \cdots\cdots(43)$$

と式変形できる．さらに続けて，式(31)と式(37)，式(32)と式(38)と対比させることにより，

$$g_0^{(L)} = 1,\ g_1^{(L)} = 1 \quad \cdots\cdots(44)$$

$$g_0^{(H)} = 1,\ g_1^{(H)} = -1 \quad \cdots\cdots(45)$$

で与えられる．

ところで，式(39)，式(40)の計算処理において，1 ブロックの出力信号 $\left\{y_{2m}, y_{2m+1}\right\}$ はそれぞれ，2 個の入力信号 $\left\{e_m^{(L)}, 0\right\}$ と $\left\{e_m^{(H)}, 0\right\}$ から計算されることに気づく．したがって，出力計算に際しては入力信号 $\left\{e_m^{(L)}, e_m^{(H)}\right\}$ のそれぞれの各サンプル値の後に 0（数値のゼロ）を補間（インタポレーション：interpolation）した後でフィルタ処理した後，再合成しなければならない．このように 0 を補間する操作を“アップサンプリング（up sampling）”といい，二つのフィルタによって分割された信号を再合成したときに，全体としてデータ数を一定に保てることになるのである．なお，アップサンプリングを実現するものを“アップサンプラ（up sampler）”という．

フィルタバンク構成とマルチレート信号処理

これまでの説明でわかるように，低域成分（ローパスフィルタの出力）と高域成分（ハイパスフィルタの出力）の二つの周波数帯域（サブバンド）に分解してDCT計算ができる．逆に，二つのサブバンド信号からもとの信号を再合成する操作は，IDCT値を計算することに等価である．

また，周波数帯域を2分割するサブバンド分解と合成は，2組のローパスフィルタとハイパスフィルタの組み合わせによって実現できるわけで，このようなシステムはフィルタバンク構成とよばれる（**図19.10**）．

図19.10において，分析フィルタバンクでは「フィルタ処理した後にダウンサンプリング」，合成フィルタバンクでは「フィルタ処理する前にアップサンプリング」を必要とする．

図19.10で“↓2”のように下向きの矢印と数字による表現は，間引き率2のダウンサンプラを表している〔**図19.11**（**a**）〕．このダウンサンプラは，2サンプルに1サンプルだけを残し，残りの1サンプルを捨てる（間引く）操作を実行する．

また，**図19.10**で“↑2”のように上向きの矢印と数字による表現は，ゼロ挿入率2のアップサンプラを表している〔**図19.11**（**b**）〕．このアップサンプラは，隣り合う2サンプルの間に信号値がゼロ(0)の信号を1サンプルずつ挿入（補間）する操作を行う．

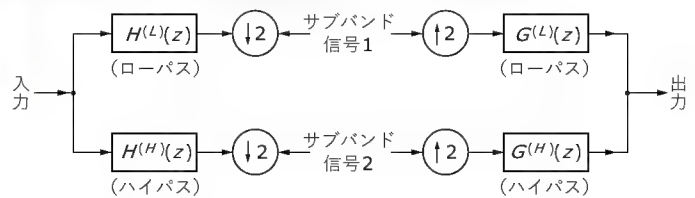
以上のようなダウンサンプラまたはアップサンプラを含む操作を“マルチレート信号処理”という．

例題2

いま，**図19.10**に示すフィルタバンク構成において，入力信号を，

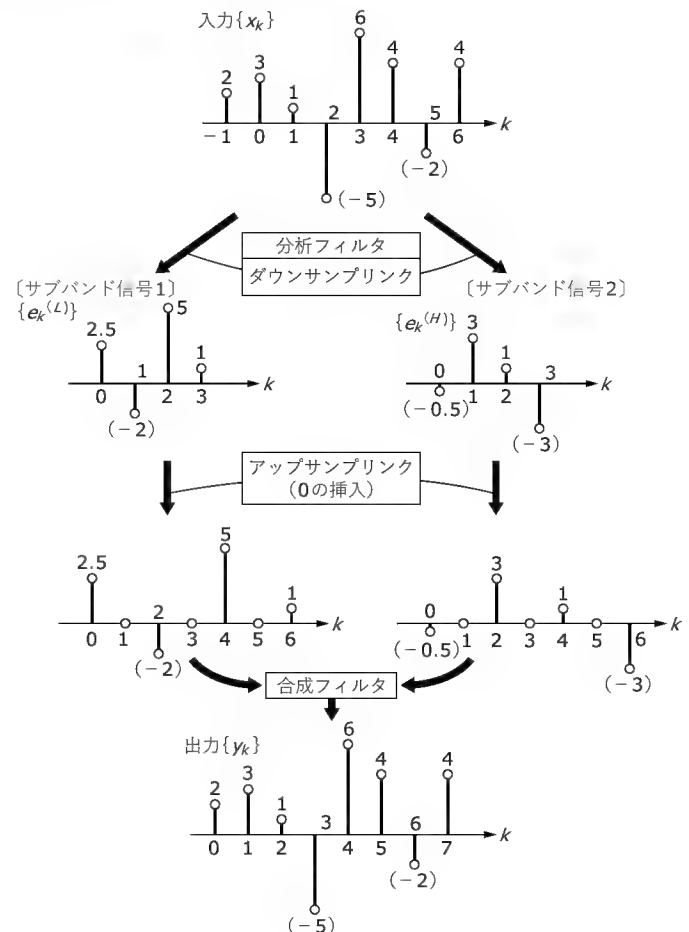
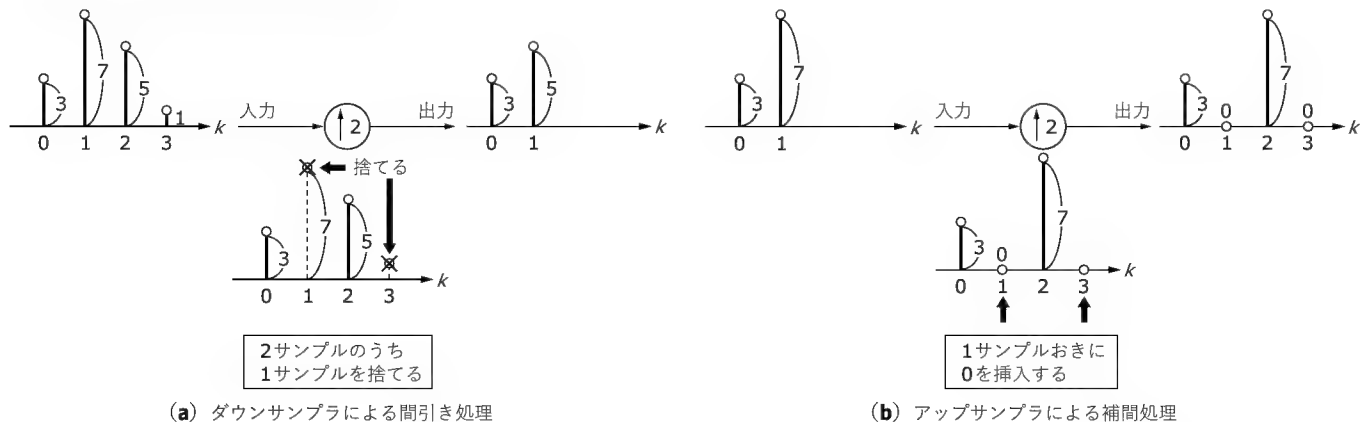
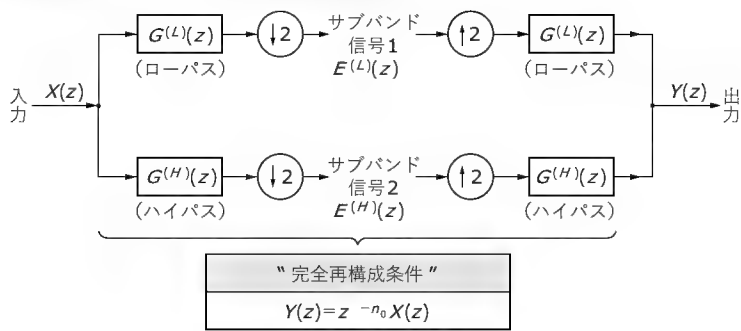
$$x_{-1}=2,\ x_0=3,\ x_1=1,\ x_2=-5,\ x_3=6,\ x_4=4,\ x_5=-2,\ x_6=4,\ \dots$$

とするとき，式(1)～式(4)に基づき，分析フィルタバンクと合成フィルタバンクの出力をそれぞれ求め，信号波形（入力，出

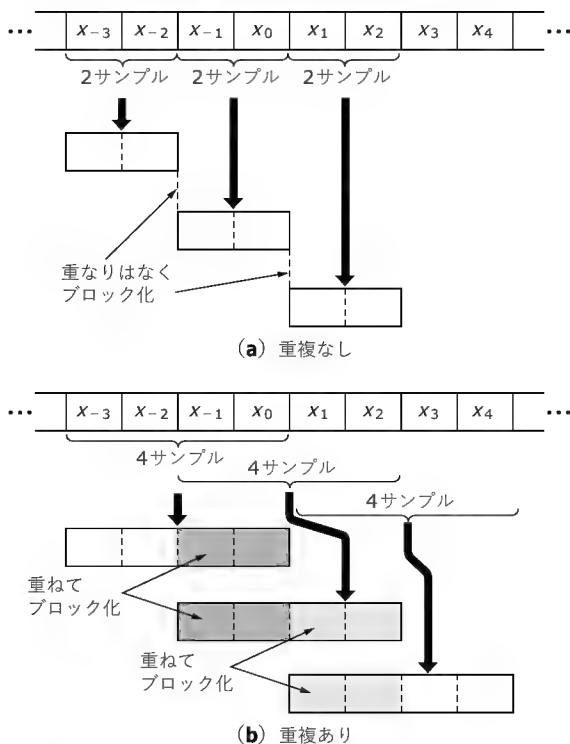
〔図19.10〕サブバンド分解/合成のフィルタバンク構成

〔図19.12〕例題2の各信号波形

〔図19.11〕ダウンサンプラとアップサンプラ

（**a**）ダウンサンプラによる間引き処理

（**b**）アップサンプラによる補間処理

〔図 19.13〕フィルタバンク構成の実現条件

〔図 19.14〕信号の計算処理の流れの相違

(a) 重複なし

(b) 重複あり

力)の概略図を示せ.

解答2

計算結果のグラフ化したものを**図 19.12**(p.195)に示しておくので，式(1)～式(4)による計算の流れを読者の皆さんには実際に計算して確認しておいてほしい.

フィルタバンク構成の実現条件

いま，DCT，IDCT 計算を2サンプルごとにブロック化するフィルタバンク構成で実現でき得るための条件を考えてみよう(**図 19.13**). この実現条件は，分析フィルタバンクの伝達関数を$H^{(L)}(z)$，$H^{(H)}(z)$とし，また合成フィルタバンクの伝達関数を$G^{(L)}(z)$，$G^{(H)}(z)$とするとき，式(7)の"完全再構成条件"を$H^{(L)}(z)$，$H^{(H)}(z)$，$G^{(L)}(z)$，$G^{(H)}(z)$で表す問題に帰着される.

ところで，ディジタル信号系列$\{r_k\}_{k=0}^{k=\infty}$のz変換$R(z)$は

$$R(z)=\sum_{k=0}^{\infty} r_k z^{-k}$$
$$=r_0+r_1z^{-1}+r_2z^{-2}+r_3z^{-3}+r_4z^{-4}+r_5z^{-5}+\cdots \quad \cdots\cdots(46)$$

で定義される. このとき，信号系列$\{r_k\}$のデータ数を半分に間引く処理，すなわちダウンサンプリングに対するz変換$R_D(z)$は，$r_1=0$，$r_3=0$，$r_5=0$，……より得られる新しい信号系列$\{r_0, r_2, r_4, \cdots\}$に対するものなので，

$$R_D(z)=r_0+r_2z^{-1}+r_4z^{-2}+\cdots \quad \cdots\cdots(47)$$

で表される. 式(47)は，

$$R_D(z)=\frac{1}{2}\Big[(r_0+r_0)+\underbrace{(r_1-r_1)}_{0}z^{-1/2}+(r_2+r_2)z^{-1} +\underbrace{(r_3-r_3)}_{0}z^{-3/2}+(r_4+r_4)z^{-2}+\cdots\Big]$$

と変形でき，さらに続けて，

$$R_D(z)=\frac{1}{2}\Big[\underbrace{\left(r_0+r_1(z^{1/2})^{-1}+r_2(z^{1/2})^{-2}+r_3(z^{1/2})^{-3}+\cdots\right)}_{R(z^{1/2})} +\underbrace{\left(r_0+r_1(-z^{1/2})^{-1}+r_2(-z^{1/2})^{-2}+r_3(-z^{1/2})^{-3}+\cdots\right)}_{R(-z^{1/2})}\Big]$$
$$=\frac{1}{2}\left\{R(z^{1/2})+R(-z^{1/2})\right\} \quad \cdots\cdots(48)$$

と表現できる.

また，信号系列$\{r_k\}$のデータ数を2倍に補間する処理，すなわちアップサンプリングに対するz変換$R_U(z)$は，ゼロ(0)値を挿入した新しい信号系列$\{r_0, 0, r_1, 0, r_2, 0\cdots\}$に対するものなので，

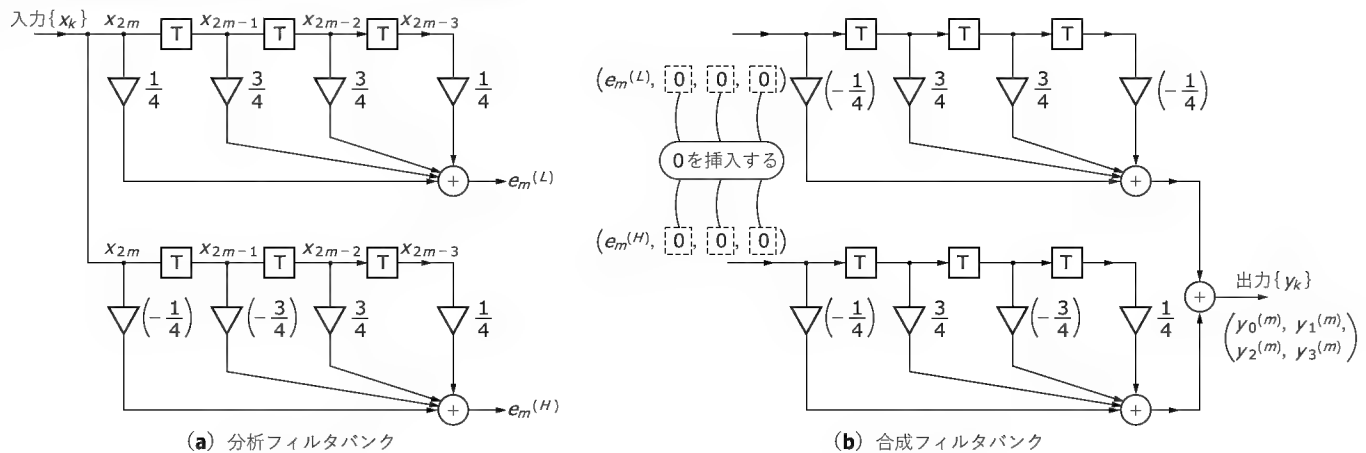
$$R_U(z)=r_0+0\cdot z^{-1}+r_1z^{-2}+0\cdot z^{-3}+r_2z^{-4}+\cdots$$
$$=r_0+r_1(z^2)^{-1}+r_2(z^2)^{-2}+\cdots$$
$$=R(z^2) \quad \cdots\cdots(49)$$

のように表される.

以上の結果に基づき，**図 19.13** において，入力信号のz変換を$X(z)$とし，式(48)の関係を利用すれば，分析フィルタバンクにおける二つの出力信号のz変換$E^{(L)}(z)$，$E^{(H)}(z)$はそれぞれ，

$$E^{(L)}(z)=\frac{1}{2}\left\{H^{(L)}(z^{1/2})X(z^{1/2})+H^{(L)}(-z^{1/2})X(-z^{1/2})\right\} \quad \cdots\cdots(50)$$

〔図 19.15〕重複ブロック化によるフィルタバンク構成

(a) 分析フィルタバンク

(b) 合成フィルタバンク

$$E^{(H)}(z)=\frac{1}{2}\left\{H^{(H)}\left(z^{1/2}\right)X\left(z^{1/2}\right)+H^{(H)}\left(-z^{1/2}\right)X\left(-z^{1/2}\right)\right\} \quad \cdots\cdots(51)$$

と表される．さらに，合成フィルタバンクからの出力信号のz変換$Y(z)$は，式(49)の関係を利用することにより，

$$Y(z)=G^{(L)}(z)E^{(L)}\left(z^2\right)+G^{(H)}(z)E^{(H)}\left(z^2\right) \quad \cdots\cdots(52)$$

となる．よって，式(50)と式(51)を式(52)に代入すれば，

$$Y(z)=\frac{1}{2}G^{(L)}(z)\left\{H^{(L)}(z)X(z)+H^{(L)}(-z)X(-z)\right\}+\frac{1}{2}G^{(H)}(z)\left\{H^{(H)}(z)X(z)+H^{(H)}(-z)X(-z)\right\}$$

となり，最終的に$X(z)$と$X(-z)$で整理して，

$$Y(z)=\frac{1}{2}\left\{H^{(L)}(z)G^{(L)}(z)+H^{(H)}(z)G^{(H)}(z)\right\}X(z)+\frac{1}{2}\left\{H^{(L)}(-z)G^{(L)}(z)+H^{(H)}(-z)G^{(H)}(z)\right\}X(-z) \quad \cdots\cdots(53)$$

となる関係が得られる．

また，式(7)の"完全再構成条件"をz変換で表すと，

$$Y(z)=z^{-1}X(z) \quad \cdots\cdots(54)$$

であり，式(53)と等置することにより，

$$H^{(L)}(z)G^{(L)}(z)+H^{(H)}(z)G^{(H)}(z)=2z^{-1} \quad \cdots\cdots(55)$$

$$H^{(L)}(-z)G^{(L)}(z)+H^{(H)}(-z)G^{(H)}(z)=0 \quad \cdots\cdots(56)$$

という関係が導かれる．たとえば，分析フィルタバンクの伝達関数を，

$$H^{(L)}(z)=\frac{1+z^{-1}}{\sqrt{2}},\ H^{(H)}(z)=\frac{-1+z^{-1}}{\sqrt{2}} \quad \cdots\cdots(57)$$

とし，また合成フィルタバンクの伝達関数を，

$$G^{(L)}(z)=\frac{1+z^{-1}}{\sqrt{2}},\ G^{(H)}(z)=\frac{1-z^{-1}}{\sqrt{2}} \quad \cdots\cdots(58)$$

とするとき，式(55)，式(56)を試しに計算してみよう．

$$\begin{aligned}&H^{(L)}(z)G^{(L)}(z)+H^{(H)}(z)G^{(H)}(z)\\&=\left(\frac{1+z^{-1}}{\sqrt{2}}\right)\times\left(\frac{1+z^{-1}}{\sqrt{2}}\right)+\left(\frac{-1+z^{-1}}{\sqrt{2}}\right)\times\left(\frac{1-z^{-1}}{\sqrt{2}}\right)\\&=\frac{1+2z^{-1}+z^{-2}}{2}+\frac{-1+2z^{-1}-z^{-2}}{2}=2z^{-1}\end{aligned} \quad \cdots\cdots(59)$$

$$\begin{aligned}&H^{(L)}(-z)G^{(L)}(z)+H^{(H)}(-z)G^{(H)}(z)\\&=\left(\frac{1-z^{-1}}{\sqrt{2}}\right)\times\left(\frac{1+z^{-1}}{\sqrt{2}}\right)+\left(\frac{-1-z^{-1}}{\sqrt{2}}\right)\times\left(\frac{1-z^{-1}}{\sqrt{2}}\right)\\&=\frac{1-z^{-2}}{2}+\frac{-1+z^{-2}}{2}=0\end{aligned} \quad \cdots\cdots(60)$$

以上の計算結果から，式(57)，式(58)のフィルタバンクは式(55)，式(56)の"完全再構成条件"を満たすことが確認できる．

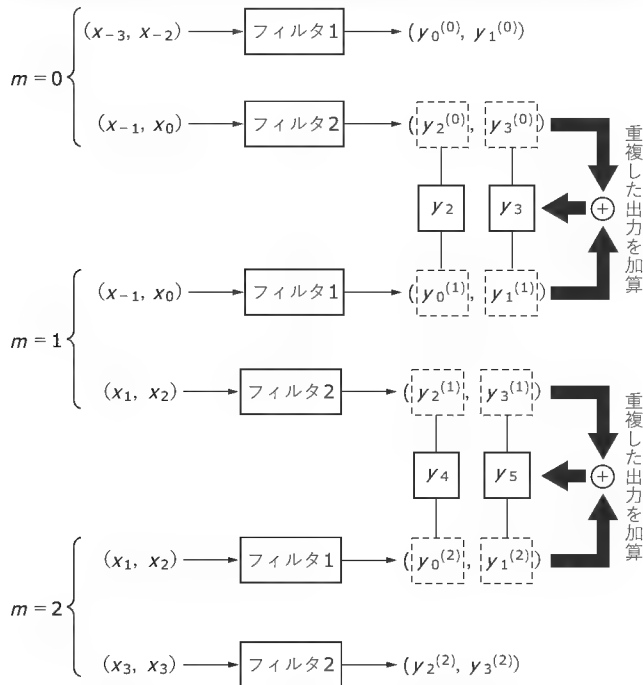
重複ブロック化によるフィルタバンク構成

これまで説明したフィルタバンク構成では，**図 19.14(a)**に示すように2サンプルを1ブロックとしてブロック同士の重なりはない形での処理だった．これに対して，**図 19.14(b)**は重なりのある形で実行するフィルタバンク構成で，各信号の計算処理の流れを示したものである．ここで，**図 19.14(b)**のフィルタバンク構成では，4サンプルを1ブロックとし，2サンプルを重複させて処理している．

いま，ディジタル信号を，

$$x_{-3},\ x_{-2},\ x_{-1},\ x_0,\ x_1,\ x_2,\ x_3,\ x_4,\ \dots$$

と左から右へと順に入力して得られる二つの周波数成分ごとに分割された信号を$e_m^{(L)}$，$e_m^{(H)}$とする．さらに周波数分割された信号$e_m^{(L)}$と$e_m^{(H)}$から再合成されて出力される信号を$y_0^{(m)}$，$y_1^{(m)}$，$y_2^{(m)}$，$y_3^{(m)}$と表し，次のように計算することを考えてみよう(**図 19.15**)．

〔図19.16〕重複ブロック化による信号処理における各信号計算の流れ

(i) $m = 0$

$$e_0^{(L)} = \frac{x_0 + 3x_{-1} + 3x_{-2} + x_{-3}}{4}$$

$$e_0^{(H)} = \frac{-x_0 - 3x_{-1} + 3x_{-2} + x_{-3}}{4}$$

$$y_0^{(0)} = -\frac{1}{4}e_0^{(L)} - \frac{1}{4}e_0^{(H)} = \frac{-6x_{-2} - 2x_{-3}}{16}$$

$$y_1^{(0)} = \frac{3}{4}e_0^{(L)} + \frac{3}{4}e_0^{(H)} = \frac{18x_{-2} + 6x_{-3}}{16}$$

$$y_2^{(0)} = \frac{3}{4}e_0^{(L)} - \frac{3}{4}e_0^{(H)} = \frac{6x_0 + 18x_{-1}}{16}$$

$$y_3^{(0)} = -\frac{1}{4}e_0^{(L)} + \frac{1}{4}e_0^{(H)} = \frac{-2x_0 - 6x_{-1}}{16}$$

上式より，出力信号のブロックの前半分($y_0^{(0)}$, $y_1^{(0)}$)と後半分($y_2^{(0)}$, $y_3^{(0)}$)がそれぞれ，入力信号の前半分(x_{-3}, x_{-2})と後半分(x_{-1}, x_0)に2分割したものから計算できることが理解される〔**図19.16(a)**〕.

(ii) $m = 1$

$$e_1^{(L)} = \frac{x_2 + 3x_1 + 3x_0 + x_{-1}}{4}$$

$$e_1^{(H)} = \frac{-x_2 - 3x_1 + 3x_0 + x_{-1}}{4}$$

$$y_0^{(1)} = -\frac{1}{4}e_1^{(L)} - \frac{1}{4}e_1^{(H)} = \frac{-6x_0 - 2x_{-1}}{16}$$

$$y_1^{(1)} = \frac{3}{4}e_1^{(L)} + \frac{3}{4}e_1^{(H)} = \frac{18x_0 + 6x_{-1}}{16}$$

$$y_2^{(1)} = \frac{3}{4}e_1^{(L)} - \frac{3}{4}e_1^{(H)} = \frac{6x_2 + 18x_1}{16}$$

$$y_3^{(1)} = -\frac{1}{4}e_1^{(L)} + \frac{1}{4}e_1^{(H)} = \frac{-2x_2 - 6x_1}{16}$$

$m = 0$のときの計算と同様に，出力信号のブロックの前半分($y_0^{(1)}$, $y_1^{(1)}$)と後半分($y_2^{(1)}$, $y_3^{(1)}$)がそれぞれ，入力信号の前半分(x_{-1}, x_0)と後半分(x_1, x_2)に2分割したものから計算できることが理解される〔**図19.16(b)**〕.

(iii) $m = 2$

$$e_2^{(L)} = \frac{x_4 + 3x + 3x_2 + x_1}{4}$$

$$e_3^{(H)} = \frac{-x_4 - 3x_3 + 3x_2 + x_1}{4}$$

$$y_0^{(2)} = -\frac{1}{4}e_2^{(L)} - \frac{1}{4}e_2^{(H)} = \frac{-6x_2 - 2x_1}{16}$$

$$y_1^{(2)} = \frac{3}{4}e_2^{(L)} + \frac{3}{4}e_2^{(H)} = \frac{18x_2 + 6x_1}{16}$$

$$y_2^{(2)} = \frac{3}{4}e_2^{(L)} - \frac{3}{4}e_2^{(H)} = \frac{6x_4 + 18x_3}{16}$$

$$y_3^{(2)} = -\frac{1}{4}e_2^{(L)} + \frac{1}{4}e_2^{(H)} = \frac{-2x_4 - 6x_3}{16}$$

このように，出力信号のブロックの前半分($y_0^{(2)}$, $y_1^{(2)}$)と後半分($y_2^{(2)}$, $y_3^{(2)}$)がそれぞれ，入力信号の前半分(x_1, x_2)と後半分(x_3, x_4)に2分割したものから計算される〔**図19.16(c)**〕.

最終的な出力は，重複を考慮して以下のように加算される(**図19.16**の点線で囲まれた変数に着目).

$$y_2 = y_2^{(0)} + y_0^{(1)}$$
$$= \frac{6x_0 + 18x_{-1}}{16} + \frac{-6x_0 - 2x_{-1}}{16} = \frac{16x_{-1}}{16} = x_{-1} \quad \cdots\cdots(61)$$

$$y_3 = y_3^{(0)} + y_1^{(1)}$$
$$= \frac{-2x_0 - 6x_{-1}}{16} + \frac{18x_0 + 6x_{-1}}{16} = \frac{16x_0}{16} = x_0 \quad \cdots\cdots(62)$$

$$y_4 = y_2^{(1)} + y_0^{(2)}$$
$$= \frac{6x_2 + 18x_1}{16} + \frac{-6x_2 - 2x_1}{16} = \frac{16x_1}{16} = x_1 \quad \cdots\cdots(63)$$

$$y_5 = y_3^{(1)} + y_1^{(2)}$$
$$= \frac{-2x_2 - 6x_1}{16} + \frac{18x_2 + 6x_1}{16} = \frac{16x_2}{16} = x_2 \quad \cdots\cdots(64)$$

以上のように，分析ディジタルフィルタでは，2分割で2サンプルを重複させて，1ブロックを4サンプルで取り出す(ダウンサンプリング)処理を実現している．一方，合成ディジタルフィルタでは，2サンプルを重複させて加算し，2サンプルの出力信号を得る(アップサンプリング)処理がなされている.

よって，式(61)～式(64)より入出力信号の相互関係式は，

$$y_k = x_{k-3} \quad \cdots\cdots(65)$$

と表され，3サンプルだけ遅れた信号として出力される．すな

わち，**図 19.3** の“完全再構成条件”($n_0=3$ に相当)を満たすことになり，フィルタバンク構成を実現できる．

以上の重複ブロック化による信号計算の流れを，m 番目のブロックに対する一般式として示せば，次のように表される．

- 分析フィルタバンクにおける計算

$$e_m^{(L)}=h_0^{(L)}x_{2m}+h_1^{(L)}x_{2m-1}+h_2^{(L)}x_{2m-2}+h_3^{(L)}x_{2m-3} \quad\cdots\cdots(66)$$

$$e_m^{(H)}=h_0^{(H)}x_{2m}+h_1^{(H)}x_{2m-1}+h_2^{(H)}x_{2m-2}+h_3^{(H)}x_{2m-3} \quad\cdots\cdots(67)$$

- 合成フィルタバンクにおける計算

$$y_0^{(m)}=g_0^{(L)}e_m^{(L)}+g_0^{(H)}e_m^{(H)} \quad\cdots\cdots(68)$$

$$y_1^{(m)}=g_1^{(L)}e_m^{(L)}+g_1^{(H)}e_m^{(H)} \quad\cdots\cdots(69)$$

$$y_2^{(m)}=g_2^{(L)}e_m^{(L)}+g_2^{(H)}e_m^{(H)} \quad\cdots\cdots(70)$$

$$y_3^{(m)}=g_3^{(L)}e_m^{(L)}+g_3^{(H)}e_m^{(H)} \quad\cdots\cdots(71)$$

最終的には，ブロックごとに重複部分の和，すなわち，

$$y_{2m}=y_2^{(m-1)}+y_0^{(m)}=x_{2m-3} \quad\cdots\cdots(72)$$

$$y_{2m+1}=y_3^{(m-1)}+y_1^{(m)}=x_{(2m+1)-3}=x_{2m-2} \quad\cdots\cdots(73)$$

として再合成された出力信号が得られる．

また，伝達関数で別表現した結果を以下に示す．

- 分析フィルタバンクの伝達関数

$$H^{(L)}(z)=h_0^{(L)}+h_1^{(L)}z^{-1}+h_2^{(L)}z^{-2}+h_3^{(L)}z^{-3} \quad\cdots\cdots(74)$$

$$H^{(H)}(z)=h_0^{(H)}+h_1^{(H)}z^{-1}+h_2^{(H)}z^{-2}+h_3^{(H)}z^{-3} \quad\cdots\cdots(75)$$

- 合成フィルタバンクの伝達関数

$$G^{(L)}(z)=g_0^{(L)}+g_1^{(L)}z^{-1}+g_2^{(L)}z^{-2}+g_3^{(L)}z^{-3} \quad\cdots\cdots(76)$$

$$G^{(H)}(z)=g_0^{(H)}+g_1^{(H)}z^{-1}+g_2^{(H)}z^{-2}+g_3^{(H)}z^{-3} \quad\cdots\cdots(77)$$

一例として，式(55)，式(56)の“完全再構成条件”を満たすフィルタバンクの伝達関数を示しておくので，時間があるときにでも検証してもらいたい．

$$H^{(L)}(z)=\frac{1+3z^{-1}+3z^{-2}+z^{-3}}{4}=\frac{\left(1+z^{-1}\right)^3}{4}$$

$$H^{(H)}(z)=-\frac{1+3z^{-1}-3z^{-2}-z^{-3}}{4}=-\frac{\left(1-z^{-1}\right)\left(1+4z^{-1}+z^{-2}\right)}{4}$$

$$G^{(L)}(z)=\frac{-1+3z^{-1}+3z^{-2}-z^{-3}}{4}=-\frac{\left(1+z^{-1}\right)\left(1-4z^{-1}+z^{-2}\right)}{4}$$

$$G^{(H)}(z)=\frac{-1+3z^{-1}-3z^{-2}+z^{-3}}{4}=-\frac{\left(1-z^{-1}\right)^3}{4}$$

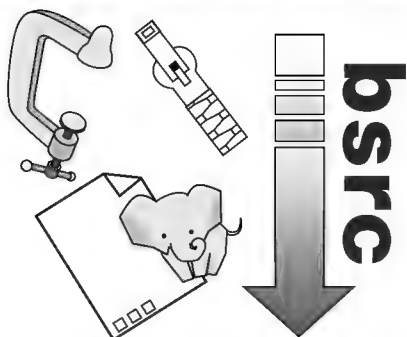
なお，入出力信号の z 変換の $X(z)$，$Y(z)$ の間には，

$$Y(z)=z^{-3}X(z) \quad\cdots\cdots(78)$$

となる関係が成立しており，式(65)の3サンプルだけ遅れて出力される“完全再構成条件”を満たすこともわかる．

*　　　*

次回は，DCTの高速計算アルゴリズムと応用例を採り上げ，わかりやすく解説していく予定である．お楽しみに．

みたに・まさあき　東京電機大学工学部情報通信工学科

ブロックソートとレンジコーダによるファイルの圧縮

高性能圧縮ツールbsrcの理論と実装(前編)

広井　誠

現在までにさまざまな汎用ファイル圧縮ツールが公開されている．今回解説するbsrcは，圧縮アルゴリズムとしてブロックソートとレンジコーダを採用し，高圧縮率で知られLinuxカーネルソースの配布などでも用いられるbzip2並みの圧縮率を実現する．

そこで本稿では，bsrcのアルゴリズムを通じて，圧縮アルゴリズムの基礎を学ぶ．なお，bsrcの全ソースリストおよび実行ファイルはWeb上で公開され，容易に入手が可能である[注]．　(編集部)

ファイルの圧縮ツールといえば，日本ではLHAが標準ツールとして一般に使用されています．このほかに，Windowsではzip，UNIXではgzipがよく使われます．最近，拡張子がbz2というファイルもよく見かけるようになりました．このファイルはbzip2[4)]というツールで圧縮されていて，一般的なファイルではLHAやgzipよりも圧縮率が高くなります．

bzip2の圧縮率が高い理由は，ブロックソート(BlockSorting)という方法を使っているからです．ブロックソートは1994年にM.BurrowsとD.J.Wheelerが提案した方法で，Burrows-Wheeler Transform(BWT)とも呼ばれています．ブロックソートの特徴は，データを圧縮するのではなく，データを圧縮しやすい形に変換することです．このあと，ほかの方法でデータを圧縮します．

ブロックソートでファイルを圧縮する場合，ブロックソートでデータを変換したあと，Move To Front(MTF)法やランレングス(Run Length)を適用してさらにデータを変換し，最後にハフマン符号や算術符号で符号化するのが一般的です．このような簡単な方法でもLHAやgzipよりファイルを圧縮できるのですが，bzip2のような高い圧縮率を達成することはできません．

MTF法やランレングスはとても簡単なアルゴリズムなので，ブロックソート向きに改良する余地は十分にあります．また，ハフマン符号の代わりにレンジコーダ(RangeCoder)という方法を使うことで，ブロックソートの圧縮率を改善することができます．これらの改良でどこまで圧縮率が向上するのか，筆者が実際にプログラム(bsrc)を作って試してみたところ，bzip2に匹敵する圧縮率を達成することができました．

本稿ではブロックソートとレンジコーダの基本を解説し，圧縮率を改善するために行った改良点について説明します．

〔図1〕ブロックソートの符号化

ブロックソート

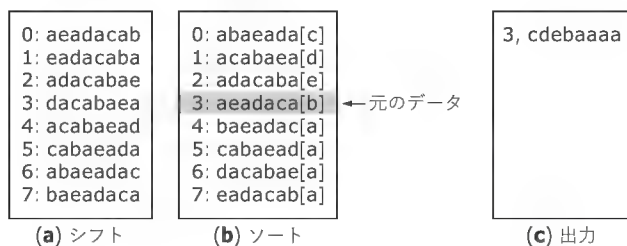
● ブロックソートの符号化

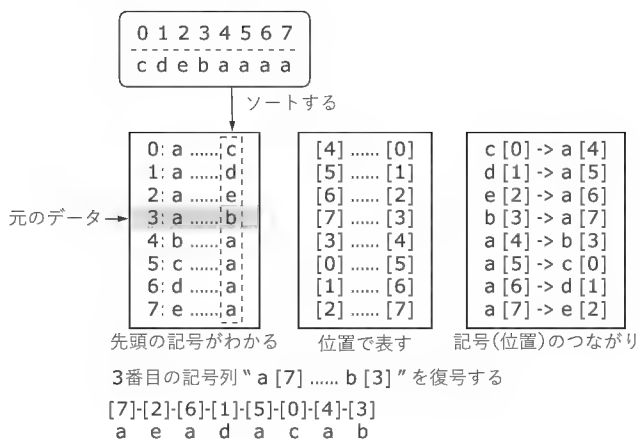
ブロックソートはその名前から，特別なソートを使うのではないかと思われた方もいるでしょう．ところが，ブロックソートというソート法があるわけではありません．ブロックソートの動作はとても簡単で，特別なソートを使わなくても実現できます．ただし，ソートは時間がとてもかかる処理なので，実用的な圧縮ツールを作成する場合，時間を短縮するための工夫が必要になります．

それでは，記号列" aeadacab "をブロックソートで変換してみましょう．まず，記号列を1記号ずつシフトして，新しい記号列を生成します．これを記号列が1回転するまで続けます．生成された記号列は**図1(a)**のようになります．

このように，生成された記号列は元のデータを含めて8個になります．次に，これらの記号列をソートします．その結果は**図1(b)**のようになります．最後に，**図1(b)**から元のデータの位置と，各記号列の最後の記号を順番に取り出して出力します．これでブロックソートは終了です．

記号列は1回転しているので，最後の記号を順番に取り出すことで記号の並びは変わりますが，記号列の中身(aが4個，b，c，d，eが1個)に変わりはありません．ようするに，ブロックソートは記号列" aeadacab "を" cdebaaaa "に変換しているだけなのです．

ここで，変換後の記号列" cdebaaaa "に注目してください．同じ記号aが並んでいることがわかります．これがブロックソートの効果です．この例では記号は8個しかありませんが，もっと長い記号列をブロックソートすれば，同じ記号を多数並べることができます．たとえば，theを多数含むテキストをブロッ

注：本稿で掲載したプログラムはhttp://www.cqpub.co.jp/interface/download/からダウンロードできる．

〔図2〕ブロックソートの復号

クソートすると，“ e.....th ”や“ he.....t ”といった記号列が生成されます．その結果，変換後の記号列には多数のhやtが並ぶことになります．

このように，ブロックソートはデータを並べ替えるだけなので，記号の出現確率はまったく変わっていないことに注意してください．したがって，このデータをそのままハフマン符号で圧縮しても効果はまったくありません．ブロックソートのあと，Move To Front法とランレングスを適用してデータを変換すると，ハフマン符号でも効率よく圧縮することができます．

● ブロックソートの復号

次はブロックソートの復号について説明します．記号列“ aeadacab ”をブロックソートで変換すると，“ cdebaaaa ”と元のデータの位置3を出力しました．この情報から元の記号列を復号します．ブロックソートの復号は，パズルを解くような，じつに面白い方法です．**図2**を見てください．

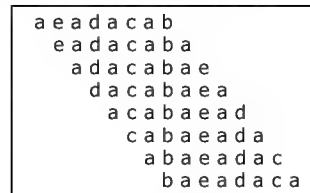
最初に，記号列“ cdebaaaa ”を記号単位でソートします．結果は“ aaaabcde ”になります．ブロックソートは，ソートした記号列の最後の記号を順番に取り出して出力しています．したがって，出力された記号列を記号単位でソートすれば，先頭の記号を求めることができます．

次に記号のつながりを求めます．ブロックソートは記号列を一つずつシフトすることで新しい記号列を生成しています．“ aeadacab ”を一つシフトすると“ eadacaba ”になります．このように，先頭の記号は末尾へ移動するので，最後の記号と先頭の記号はつながっていることがわかります．

つまり，記号bの次は記号aになります．同じ記号が複数あるので，記号は位置で表したほうがわかりやすいでしょう．この場合，記号bの位置は3になります．3の次は位置7の記号aで，7の次は2の記号eというように，記号のつながりを求めることができます．

記号のつながりがわかっても，どこから復号を始めたらいいかわかりません．このために，元のデータの位置を出力しています．

〔図3〕記号列のソート

```
記号列 a e a d a c a b a e a d a c a b
                       ~~~~~~~~~~~~~~~
                           コピー

a e a d a c a b
  e a d a c a b a
    a d a c a b a e
      d a c a b a e a
        a c a b a e a d
          c a b a e a d a
            a b a e a d a c
              b a e a d a c a

index [0 1 2 3 4 5 6 7]
各記号の位置を index にセットしてソートする
```

〔リスト1〕ブロックソートの符号化と復号

```
typedef unsigned char   Uchar;

/* バッファ定義 */
Uchar buffer1[BUF_SIZE * 2];
Uchar buffer2[BUF_SIZE];
int   index[BUF_SIZE];

/* 符号化 */
int blocksort_encode( Uchar *out, Uchar *in, int size )
{
  int i, top;
  memcpy( in + size, in, size );  /* コピー */
  sort_e( in, size );             /* ソート */
  /* 出力 */
  for( i = 0; i < size; i++ ){
    int n = index[i];
    if( n == 0 ) top = i;
    out[i] = in[n + size - 1];
  }
  return top;
}

/* 復号 */
void blocksort_decode( Uchar *out, Uchar *in, int size,
                                                  int top )
{
  int i, n;
  sort_d( in, size );    /* 1 文字でソート */
  /* 出力 */
  n = index[top];
  for( i = 0; i < size; i++ ){
    out[i] = in[n];
    n = index[n];
  }
}
```

この場合，元データの位置は3ですから，3番目の記号列“ a.....b ”を復号します．記号aの位置7から記号のつながりをたどっていけば，元の記号列“ aeadacab ”を復号することができます．

ブロックソートの場合，符号化にはソートが必要になるため時間がかかりますが，復号はとても簡単に行うことができます．

● ブロックソートの実装

ブロックソートをプログラムする場合，文字列のソートのように記号の位置を配列に格納して，その配列をソートすることで実現できます．このとき，**図3**に示すように同じ記号列を後ろへコピーすると，プログラムを簡単に作ることができます．**リスト1**を見てください．

データをbufferに読み込み，**図3**のように各記号の位置をindexにセットします．そして，indexを使ってソートを行います．あとの処理はとても簡単です．

ブロックソートで問題になるのが符号化のときのソートです．

記号列が長くなると，クイックソートでもとても時間がかかります．時間を短縮するため，先頭2記号で分布数えソートを行い，そのあとでマージソートやクイックソートを行うなど，いろいろな工夫が考えられますが，もっと高速にソートできる方法があります．

巨大なテキストデータを高速に検索するためのデータ構造にsuffix arrayがあります．suffix arrayの構成にはブロックソートと同様にデータのソートが必要になるため，今までに高速なアルゴリズムが研究・開発されています．これらのアルゴリズムをブロックソートに適用することで，巨大なデータでも比較的短時間でソートすることができます．

suffix arrayはデータの終端を考慮してソートするので，ブロックソートでも記号列に終端記号を付加してソートする方法があります．この場合，単純な文字列の比較ではソートできないので，終端記号に対応した比較関数を作成しなければなりません．そして，終端記号を含めてブロックソートするように処理を変更します．

本稿のプログラム(bsrc)は記号列に終端記号を付加していませんが，suffix arrayのアルゴリズムを参考に作成しています．オプションを指定しないと二段階ソート法を参考にしたマージソート，オプション-sを指定するとLarsson，Sadakane(定兼)法でソートします．一般的なデータでは二段階ソート法のほうが高速ですが，筆者の実装では繰り返しの多いデータだと極端に遅くなることがあります．このようなデータでもLarsson，Sadakane法を使うと高速にソートできます．

これらのアルゴリズムはとても興味深いのですが，圧縮率の改善が本稿の趣旨なので説明は割愛します．参考URL5)にはLarsson，Sadakane法を説明したスライドが掲載されています．また，参考URL6)にはsuffix arrayを使用した全文検索システムissのソースが公開されています．興味のある方はこれらのWebページを参照してください．

〔図4〕MTFによる符号化

```
baccdddd [a, b, c, d] ──► [b, a, c, d] MTF : 1
*                  * b を先頭に移動

baccdddd [b, a, c, d] ──► [a, b, c, d] MTF : 11
 *                 * a を先頭に移動

baccdddd [a, b, c, d] ──► [c, a, b, d] MTF : 112
  *                  * c を先頭に移動

baccdddd [c, a, b, d] ──► [c, a, b, d] MTF : 1120
   *             * c を先頭に移動

baccdddd [c, a, b, d] ──► [d, c, a, b] MTF : 11203
    *                * d を先頭に移動

baccdddd [d, c, a, b] ──► [d, c, a, b] MTF : 112030
     *           * d を先頭に移動

同じことを繰り返す．baccdddd => 11203000
```

Move To Front 法

次はMove To Front(MTF)法について説明します．MTFは同じ記号がいくつ前に現れたかを符号にする方法で，最近現れた記号ほど小さな値に変換することができます．MTFは記号の出現表を作ることで簡単に実現できます．たとえば，記号の種類が{a，b，c，d}の四つしかない場合で，記号列“abccddddd”をMTFで符号化してみましょう．**図4**を見てください．

まず，表を[a，b，c，d]に初期化します．MTFはこの表に現れる記号の位置を符号にします．ここでは先頭を0で表すことにします．最初の記号bの位置は1番目なので1を出力します．ここで，記号bを表の先頭へ移動します．このように，記号を表の先頭へ移動することからMove To Frontと呼ばれています．この結果，頻繁に現れる記号は表の先頭付近に集まるので，それらの記号を小さな値に変換することができます．

MTFにはもう一つ特徴があります．記号dを変換するところを見てください．最初のdは表の3番目にあるので3を出力します．そして，dを表の先頭へ移動するので，残りのdは三つともすべて0に変換されます．このように，MTFでは同じ記号が連続していれば，それを0に変換することができます．この特徴がブロックソートに適しているのです．

ブロックソートで変換されたデータは同じ記号が多数並びます．このデータをMTFで変換すれば，同じ記号の並びを0に変換することができます．つまり，tが並んでいるところも，hが並んでいるところも0に変換されるのです．したがって，データ全体の中で0の割合が著しく増加し，記号の出現確率を大きく偏らせることができるので，ハフマン符号や算術符号でも効率よく圧縮できるのです．

● MTF法の改良

MTF法はその名前が示すように，現れた記号を表の先頭へ移動します．この移動位置を変更することで，ブロックソートの圧縮率を改善できます．本稿のプログラム(bsrc)では移動位置を2番目に変更しています．そして，先頭へ移動できるのは，2番目にある記号の場合だけに限定します．つまり，いきなり先頭へ移動するのではなく，まず2番目に移動しておいて，それから先頭へ移動するのです．

たとえば，記号がa，b，cの3種類で，記号列“aaaacbaaaa”をMTFで変換してみましょう．単純なMTFは**図5**のように変換されます．

このように，aの途中でbとcが入っている場合，bとcを先頭へ移動すると“0000222000”に変換されます．それでは，bとcを2番目に移動させてみましょう．**図6**を見てください．

bとcを2番目に移動すると，“aaaacbaaaa”は“0000220000”に変換されます．2番目に移動することで0を一つ増やすことができました．長い記号列をブロックソートで変換すると同じ

〔図5〕単純なMTFによる符号化

```
aaaacbaaaa [a, b, c] ──→ [c, a, b] MTF：00002
    *             * c を先頭に移動

aaaacbaaaa [c, a, b] ──→ [b, c, a] MTF：000022
     *            * b を先頭に移動

aaaacbaaaa [b, c, a] ──→ [a, b, c] MTF：0000222
      *           * a を先頭に移動

aaaacbaaaa => MTF：0000222000
```

〔図6〕改良版MTFによる符号化(MTF2)

```
aaaacbaaaa [a, b, c] ──→ [a, c, b] MTF2：00002
    *             * c を 2 番目に移動

aaaacbaaaa [a, c, b] ──→ [a, b, c] MTF2：000022
     *            * b を 2 番目に移動

aaaacbaaaa [a, b, c] ──→ MTF2：0000220
      *       * a は先頭のまま

aaaacbaaaa => MTF2：0000220000
```

〔図7〕MTF2の弱点

```
aaaabcaaaa [a, b, c] ──→ [b, a, c] MTF2：00001
    *           * b を 1 番目に移動

aaaabcaaaa [b, a, c] ──→ [b, c, a] MTF2：000012
     *            * c を 2 番目に移動

aaaabcaaaa [b, c, a] ──→ [b, a, c] MTF2：0000122
      *           * a を 2 番目に移動

aaaabcaaaa [b, a, c] ──→ [a, b, c] MTF2：00001221
       *        * a を 1 番目に移動

aaaabcaaaa => MTF2：0000122100
```

〔図8〕1番目の記号の移動を制限する

```
aaaabcaaaa [a, b, c] ──→ [a, b, c] MTF3：00001
    *             * 直前に出力した記号は 0 なので
                    b は移動しない

aaaabcaaaa [a, b, c] ──→ [a, c, b] MTF3：000012
     *            * c を 2 番目に移動

aaaabcaaaa [a, b, c] ──→ MTF3：0000120
      *       * a は先頭

aaaacaaaa => MTF3：0000120000
```

記号が多数並びますが，連続した記号の間に他の記号が入ることも多くあるはずです．したがって，このMTF法で変換すれば，0の個数を増やすことが期待できます．

ところが，2番目に移動する方法(MTF2)にも弱点があります．**図7**を見てください．

記号が{a, b, c}の3種類で，記号列“aaaabcaaaa”をMTF2で変換します．この場合，aが続いたあとの記号bが2番目にあるため，bを先頭へ移動します．そして，cを2番目に移動するので，aの位置は3番目になります．このあとaが続くと，aは2番目に移動してから先頭に戻るので，結果は“0000122100”になります．これでは0の個数が減ってしまいます．

そこで，直前に出力した記号が0以外の場合にのみ，1番目の記号を先頭へ移動することにします．つまり，0を出力した直後は，1番目の記号を移動しないのです．この方法で“aaaabcaaaa”を変換すると，**図8**のようになります．

このように，“aaaabcaaaa”は“0000120000”に変換されます．MTF2よりも0の個数を増やすことができました．この改良はURL[7]を参考にしました．MTF法のプログラムはとても簡単です．詳細は**リスト2**をお読みください．

ランレングス

ランレングスとは「連続して現れるものの長さ」という意味で，データ内で同じ値が並んでいる場合はその値と個数で符号化する方法のことを，「ランレングス圧縮」または「ランレングス符号化」といいます．ランレングスはとても簡単な符号化ですが，

〔リスト2〕Move To Front法

```
/* 符号化 */
void mtf_encode( Uchar *out, Uchar *in, int size )
{
  int i, prev = 1;
  Uchar mtf_table[256];
  for( i = 0; i < 256; i++ ) mtf_table[i] = i;
  for( i = 0; i < size; i++ ){
    int c = *in++;
    int j = 0;
    while( mtf_table[j] != c ) j++;
    if( j == 1 ){
      if( prev ){
        mtf_table[1] = mtf_table[0];
        mtf_table[0] = c;
      }
    } else if( j > 1 ){
      memmove( mtf_table + 2, mtf_table + 1, j - 1 );
      mtf_table[1] = c;
    }
    *out++ = j;
    prev = j;
  }
}

/* 復号 */
void mtf_decode( Uchar *out, Uchar *in, int size )
{
  int i, prev = 1;
  Uchar mtf_table[256];
  for( i = 0; i < 256; i++ ) mtf_table[i] = i;
  for( i = 0; i < size; i++ ){
    int j = *in++;
    int c = mtf_table[j];
    if( j == 1 ){
      if( prev ){
        mtf_table[1] = mtf_table[0];
        mtf_table[0] = c;
      }
    } else if( j > 1 ){
      memmove( mtf_table + 2, mtf_table + 1, j - 1 );
      mtf_table[1] = c;
    }
    *out++ = c;
    prev = j;
  }
}
```

それでもいくつかの方法が考えられます．いちばん簡単な方法は，データの値とデータの個数で表す方法です．たとえば，“aabbbcccc”の記号列は[a，2，b，3，c，4]と表すことができます．

ところが，この方法ではブロックソートの圧縮率を改善することはできません．また，データが連続しているところだけを符号化するために，ランレングスの開始記号を定義する方法がありますが，この方法でも効果はまったくありません．いろいろ試してみたところ，簡単で効果的な方法が二つありました．

一つは，データ0だけをランレングスで符号化するという方法です．これを「ゼロランレングス」といいます．この方法では，0をランレングスの開始記号にすることができるので，「0＋データの個数」のように2バイトで符号化することができます．MTF法で変換したデータは0の個数がきわめて多くなるので，効率よく圧縮できます．

もう一つは，同じデータが数個以上続いていたらランレングスで符号化するという方法です．たとえば，同じデータが三つ以上続いていたら符号化することにしましょう．すると，データが“aaaaa”の場合は[a，a，a，2]と符号化されます．逆に，“aaa”は[a，a，a，0]と符号化されるので1バイト増えることになりますが，連続していないデータをランレングスで符号化することはないので，単純なランレングスよりもデータが伸張する危険性は小さくなるはずです．そして，0は長く連なっている場合が多いので，ランレングスでも効率よく圧縮することができます．

● ランレングスの改良

どちらの方法でも圧縮率は大きく向上するのですが，ゼロランレングスを改良したZero Length Encoding[7]を使うと圧縮率をさらに改善することができます．この方法はゼロランレングスと同様に記号0のみをランレングスで符号化しますが，このとき0と1を使って記号0の個数を2進数で表すところがポイントです．つまり，1ビットを1バイトで表して，個数を0と1の記号列で表すのです．0と1を使って個数を表すので，ほかの記号も変換が必要になります．

数を2進数で表す場合，最上位ビットは常に1になるので省略することができます．Zero Length Encodingでは，個数N

〔表1〕Zero Length Encoding 個数Nの符号化

N	$N+1$	符　号
1	2(10)	0
2	3(11)	1
3	4(100)	0 0
4	5(101)	0 1
5	6(110)	1 0
6	7(111)	1 1
7	8(1000)	0 0 0
8	9(1001)	0 0 1
9	10(1010)	0 1 0
10	11(1011)	0 1 1

〔表2〕Zero Length Encoding 0x01～0xFFの符号化

記　号	符　号
0x01～0xfd	＋1(0x02～0xfe)
0xfe	0xff，0x00
0xff	0xff，0x01

〔リスト3〕Zero Length Encoding

```
/* 符号化 */
int zle_encode( Uchar *out, Uchar *in, int size )
{
  Uchar *wp = out;
  Uchar *limit = in + size;
  int c, count;
  while( in < limit ){
    c = *in++;
    switch( c ){
    case 0:
      /* 0 を数える */
      count = 1;
      while( in < limit && *in == 0 ){
        count++;
        in++;
      }
      /* 符号を出力 */
      count++;
      while( count != 1 ){
        *wp++ = count & 0x01;
        count >>= 1;
      }
      break;
    case 0xfe:
      *wp++ = 0xff;
      *wp++ = 0x00;
      break;
    case 0xff:
      *wp++ = 0xff;
      *wp++ = 0x01;
      break;
    default:
      *wp++ = c + 1;
    }
  }
  return wp - out;
}
```

```
/* 復号 */
int zle_decode( Uchar *out, Uchar *in, int size )
{
  Uchar *wp = out;
  Uchar *limit = in + size;
  while( in < limit ){
    if( *in <= 0x01 ){
      /* 0 と 1 を探す */
      Uchar *p;
      int i = 0, count = 1;
      do{
        i++; in++;
      } while( in < limit && *in <= 0x01 );
      /* 数値に変換する */
      for( p = in - 1; i > 0; i-- ){
        count = (count << 1) + *p--;
      }
      count--;
      /* 出力 */
      while( count-- > 0 ) *wp++ = 0;
    } else {
      int c = *in++;
      if( c == 0xff ){
        c = *in++;
        if( c == 0 ){
          *wp++ = 0xfe;
        } else {
          *wp++ = 0xff;
        }
      } else {
        *wp++ = c - 1;
      }
    }
  }
  return wp - out;
}
```

に1を加えて最上位以外のビットを出力します．たとえば，1から10までの値は**表1**のように変換されます．0と1は個数の符号に使うため，ほかのデータは**表2**のように変換します．

一般的には，普通のゼロランレングスのほうが効率よく圧縮できると考えられます．ところがMTF法で変換したあと，記号0は長く連続しているところだけではなく，長さが1または2のように短いところも多くあるはずです．長さが1の場合，普通のゼロランレングスでは2バイトに増えてしまいます．また，長さが2の場合も圧縮することはできません．

ところがZero Length Eoncodingでは，長さが1または2の場合でも1バイトで表すことができます．0xfeと0xffは2バイトに増えますが，MTF法で変換したあと0xfeと0xffの個数はとても少なくなるはずです．したがって，普通のゼロランレングスよりもデータが伸張する危険性は小さくなり，圧縮率の向上が期待できるというわけです．

Zero Length Eoncodingのプログラムは簡単です．詳細は**リスト3**をお読みください．本稿のプログラム（bsrc）ではZero Length Encodingをベースにして，記号0x01～0xfdは三つ以上続いていたらランレングスで符号化する処理を追加して使用しています．

*　　　　　*

今回はブロックソートについて解説しました．次回は，引き続きレンジコーダの解説を行います．

参考文献とURL

1）植松友彦，『文書データ圧縮アルゴリズム入門』，CQ出版社，1994
2）奥村晴彦，『C言語による最新アルゴリズム事典』，技術評論社，1991
3）奥村晴彦，「データ圧縮の基礎から応用まで」，『C MAGAZINE』，2002年7月号，ソフトバンク
4）The bzip2 and libbzip2 home page，http://sources.redhat.com/bzip2/
5）大規模テキスト索引（suffix array）の構築法とその情報検索への応用 suffix array構築アルゴリズムと実装，http://www.gi.k.u-tokyo.ac.jp/ssr-homepage/1999/workshop1/sadakane/
6）The iss Homepage，http://www-imai.is.s.u-tokyo.ac.jp/~sada/iss/
7）bwtzip，http://stl.caltech.edu/bwtzip.shtml
8）Zzip，http://debin.org/zzip/
9）szip homepage，http://www.compressconsult.com/szip/
10）Canterbury Corpus，http://corpus.canterbury.ac.nz/
11）M.Hiroi's Home Page，http://www.geocities.co.jp/SiliconValley-Oakland/1680/

ひろい・まこと

オープンソースのITRON仕様OS

TOPPERSで学ぶRTOS技術

第3回 ダイナミックローディング対応μITRON TOPPERS/IDLとTOPPERS開発環境

河合孝夫

TOPPERS/IDL開発の背景

● ネットワークの普及

いままで組み込み機器では，ソフトウェアが小規模であったため，品質・信頼性を確保することが比較的容易でした．また，一度出荷された機器のソフトウェアは基本的には変更しないことが前提でした．ところが最近では，とくに携帯電話機で顕著に見られるように，機器の多機能化・ネットワーク化により，組み込まれているソフトウェアの大規模化および複雑化が急速に進み，品質・信頼性の確保が問題になっています．

● 1リンクモデルだったμTRON仕様OS

いままでの組み込み機器では，ソフトウェアを最小限のハードウェア資源で効率良く動かすことに主眼をおいたリアルタイムOS(RTOS)が使われてきました．このRTOSとして国内で大きなシェアを占めているのがITRON仕様OSです．ITRON仕様においては，組み込みソフトウェアを更新するローダ機能は仕様策定範囲に含めていませんでした．

これは，いままで組み込み機器は，人がROMにOSとアプリケーションプログラムをリンクした専用プログラムを書き込み，ROMでプログラムが実行されるという前提があったからです．このROMに書かれたプログラムはROM交換以外にソフトウェアを更新する手段を持ち合わせていませんでした．しかしフラッシュメモリデバイスの低価格化，高集積化にともない，組み込み機器においてもOS自身によるソフトウェアのバージョンアップが可能となってきました．

● バージョンアップの必要性

新機種モデルや新製品をいち早く市場に投入したいというビジネス面の時間的要求は，ますます厳しくなってきています．そのため，一定の品質基準・機能完成度をもって出荷することができなくなってきています．このような状況下で一度出荷した機器に不具合があった場合，機器を回収してソフトウェアの修正を行わねばならず，大きな保守コストが必要になります．

さらに，来たるべきユビキタス時代においては，マイクロコンピュータを使い，通信・ネットワーク機能をもったさまざまな機器が家庭，オフィス，工場などあらゆるところに出現することが想像できます．「どこでもコンピューティング」環境においては，機器が相互にネットワーク接続されていることから，ユーザーが機器の購入後にソフトウェアのバージョンアップをすることが必要不可欠となると考えました．

基本スキームについて

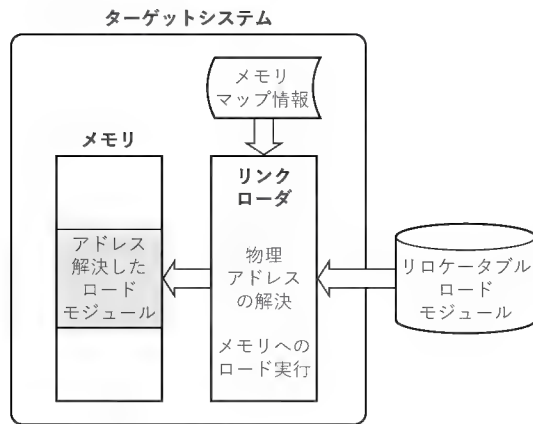
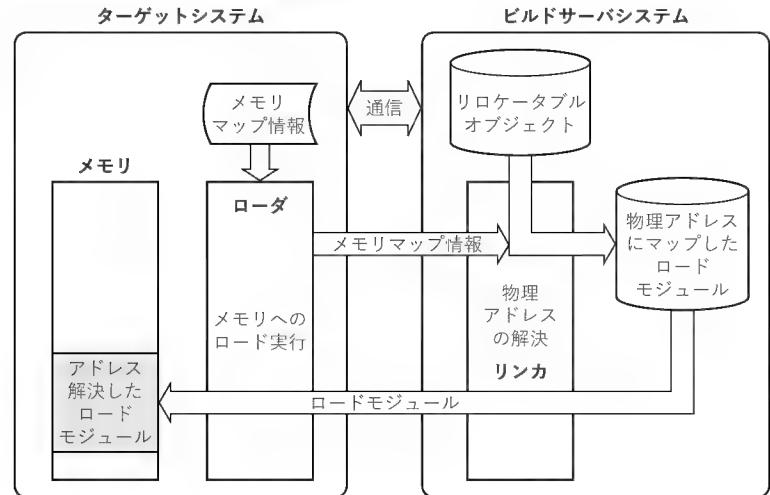
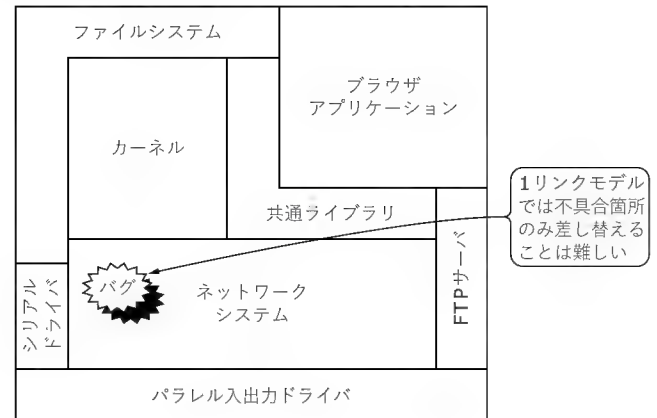
ダイナミックローディング機能をITRONで初めて実現したのが，TOPPERS/IDLカーネルです．TOPPERS/IDLカーネルは，機器に組み込まれているソフトウェアの不具合または機能更新・追加が必要な場合に，機器ユーザーがネットワーク機能を使って部分的にモジュールをダウンロードし，バージョンアップする機能をOSレベルで実現するものです．目的としては，今後の携帯電話機や無線通信技術を応用したユビキタス機器に対応する新しい保守用機能を提案すること．また，現在の組み込みソフトウェアの不具合対策および品質問題に対して，即効的な解決策を提供することを目的としました．

TOPPERS/IDLカーネルでは，メモリ管理の手法として，すべてのプログラムで単一の物理アドレス空間を共有し，実行位置に依存したプログラムを動的にロード実行するしくみを採用しました．これにより，多くのプロセッサ，組み込みシステムに適応することが可能であると考えました．

このような単一アドレス空間に実行位置依存のプログラムをロードし実行するための古典的な手法として，プログラム実行時にモジュールのアドレス解決をしながらメモリにローディングする方法があります(**図1**)．

しかし，ITRONを実行する組み込み機器を想定すると，リンカ機能を機器側に搭載するために実装メモリを拡張することは許されないと考えました．また，ローディング時にも基幹システム機能を止めないで動作させることを想定した場合，機器側の処理オーバヘッドも気になります．そこでリンクローダ機能のリンク機能は汎用OSが実行するサーバ側へ，ローダ機能のみ機器側へと機能を分離し，通信セッションにおいて機器-サーバの協調動作でアドレス解決を行う方式を採用しました(**図2**)．

次に具体的なダウンロードにおける，機器-サーバ間の協調動作のようすを説明します．

〔図1〕リンクローダ機能

〔図2〕リンカとローダ機能を分離

1) 機器はダウンロード実行時，機器のID番号，取得したいモジュールのID番号，バージョン番号，および機器のメモリマップ情報をサーバに送る
2) サーバは機器ID，ロードモジュールID，バージョン番号で生成するロードモジュールを特定する．通知されたメモリマップ情報より機器の空きメモリを検索し，ロードモジュールの各メモリセクションに対して，静的に物理アドレスを割り付けたロードモジュールファイルを生成する．このとき，サーバはロードモジュールが必要としているカーネル資源の生成情報もロードモジュールヘッダ情報として生成する
3) ターゲット機器ではロードモジュールおよびヘッダ情報ファイルを取得し，実行コードを不揮発メモリに書き込む．また，ロードモジュールをITRON処理単位として活性化するために，必要なカーネルオブジェクト(タスク，セマフォなど)の生成，メモリオブジェクトの登録を行う
4) 機器はダウンロード実行時，機器のID番号，取得したいモジュールのID番号，バージョン番号，および機器のメモリマップ情報をサーバに送る
5) サーバは機器ID，ロードモジュールID，バージョン番号で生成するロードモジュールを特定する．通知されたメモリマップ情報より機器の空きメモリを検索し，ロードモジュールの各メモリセクションに対して，静的に物理アドレスを割り付けたロードモジュールファイルを生成する．このとき，サーバはロードモジュールが必要としているカーネル資源の生成情報もロードモジュールヘッダ情報として生成する
6) ターゲット機器ではロードモジュールおよびヘッダ情報ファイルを取得し，実行コードを不揮発メモリに書き込む．また，ロードモジュールをITRON処理単位として活性化するために，必要なカーネルオブジェクト(タスク，セマフォなど)の生成，メモリオブジェクトの登録を行う

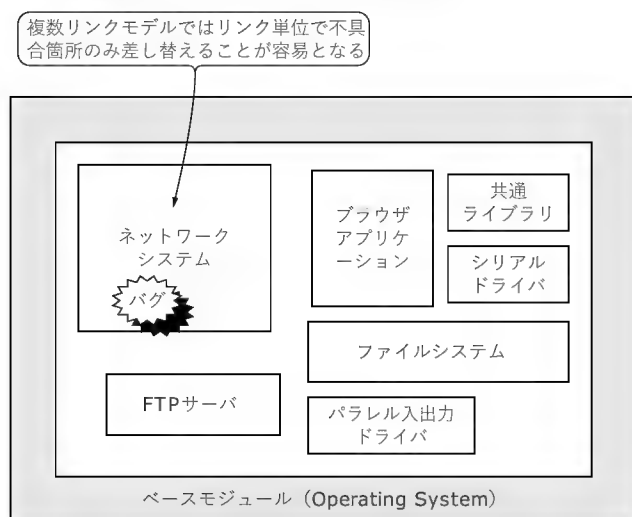
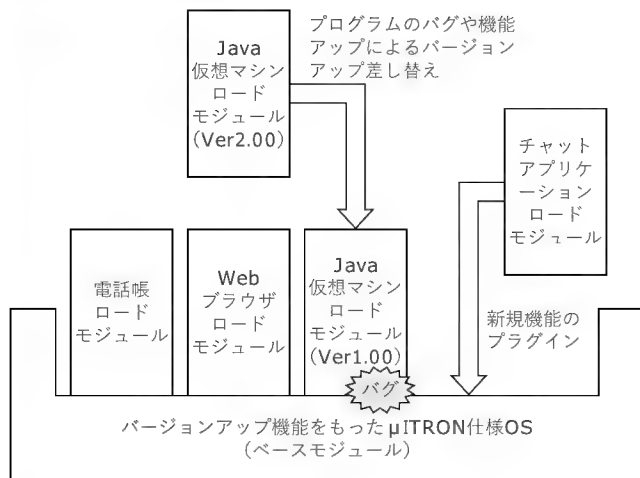
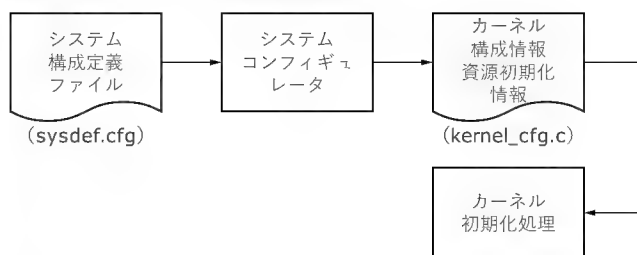
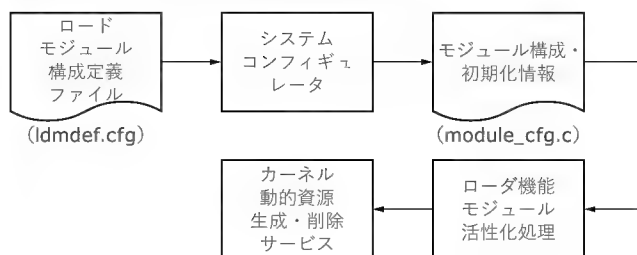
〔図3〕単一リンクモデルによるシステム構成

詳細機能の説明

● モジュール

従来のITRONにおけるシステムでは，**図3**のようにOSを含むすべてのソフトウェアが1枚岩でリンクされた形でロードモジュールが作られていました．この場合，不具合箇所のみを修正したとしても，修正による影響はロードモジュール全体に波及し，ソフトウェアをバージョンアップするためには全体を書き換えする方法しかありません．TOPPERS/IDLカーネルでは，**図4**に示すようなロードモジュールというリンク単位でソフトウェアの部分更新や，機能プラグインを行う，複数リンクモデルを採用することで，ダイナミックローディングを実現しました．

TOPPERS/IDLではOSの基幹機能が含まれているベースモジュールと，ダイナミックにロードとアンロードができるロードモジュールという二つのモジュールを定義しました(**図5**)．ベースモジュールは，いわゆるOSです．別のいい方をすれば

〔図4〕複数リンクモデルによるシステム構成

〔図5〕モジュールの概念

〔図6〕従来のシステム構成情報定義方法

〔図7〕ロードモジュール構成情報定義方法

ロードモジュールを動的に装着するための受け皿といえます．この性質上，ベースモジュールは工場出荷時に機器に組み込まれ，ダウンロードによる自身の更新はできないものとします．ロードモジュールはアプリケーションプログラムです．ロードやアンロードの対象となるので，機能単位で作成する必要があります．

● ロードモジュールの作成方法

ベースモジュールはシステムの基幹機能（カーネル，ローダ）が1リンク（システムを構成するすべてのプログラム部品を，一つの最終オブジェクトファイルに連結しアドレス解決すること）され，工場出荷時に不揮発メモリに書き込まれるものです．ベースモジュールの作成方法は，従来のITRONのシステム構築手法と同じ手法で可能であると考えました．一方，動的に結合されるロードモジュールの作成手法については，現在のITRONでは方法が確立されていません．ローディング機能を実現するとともに，どのようにロードモジュールを開発者に作成させるかの検討が必要でした．

従来のITRONシステムにおいて，開発者はシステム構成を設定する際，システムコンフィギュレーションファイルを作成し，ITRON仕様で標準化されている静的APIで構成情報（タスクや使用するセマフォなどの構成）を記述することが可能です．また，ITRON仕様のOSベンダはこの構成定義ファイルを解釈し，カーネルに対して資源を生成するための情報を生成するシステムコンフィギュレータをツールとして提供しています（**図6**）．ここで，ロードモジュールの構成定義を，従来のシステム構成編集方法と整合させることを検討しました．具体的には開発者に従来ITRON仕様と同じ静的API書式で，モジュールの構成情報を記述させることです（**リスト1**）．

また，ツールとしてロードモジュールの構成を定義したファイル（モジュールコンフィギュレーションファイル）を解釈し，ロードモジュールが必要としている資源の生成情報を機器のローダ機能に出力するモジュールコンフィギュレータを開発しました（**図7**）．

リスト1は，ロードモジュールのタスクを生成するμITRON/PX仕様の静的APIの一例です．この記述方法は，システムコンフィギュレーションファイルに記述する内容とまったく同じ書式です．システムコンフィギュレーション情報はカーネル自身に出力されるのに対して，モジュールコンフィギュレーション情報は，ローダ機能に出力されます．ローダ機能はロードモジュールを活性化する際，この構成情報を参照し，ITRONカー

〔リスト1〕モジュールコンフィギュレーション記述例（タスクの生成）

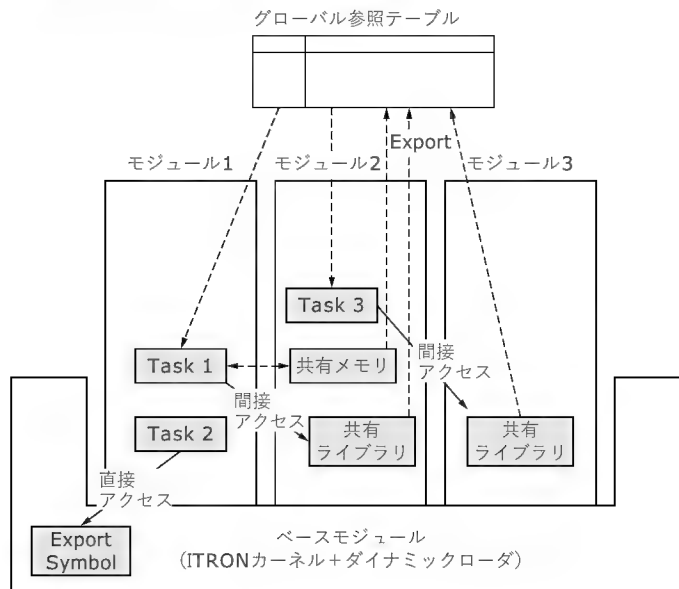
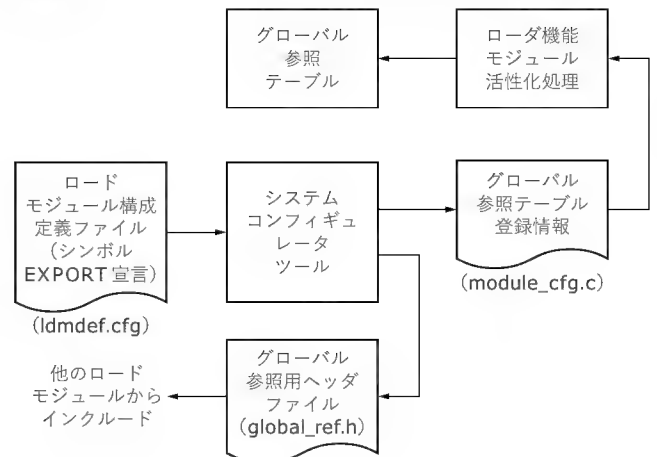
```
user_domain DOM2 {
  CRE_TSK (
    TSKID_DEMO,
    { TA_HLNG|TA_ACT, NULL,
    tsk_demotsk, 12,
    4096, NULL, 4096, NULL } ) ;
}
```

〔リスト2〕シンボルエクスポート静的API

```
/* モジュールエクスポートシンボル */
EXPORT ("char glob_flags") ;
EXPORT ("int get_status ( int device_id, char *info ) ") ;
```

〔リスト3〕global_ref.hに自動生成されたエクスポートシンボルの参照マクロ

```
#define TGR_get_status   0
#define get_status  (*  (int  (*) ( int device_id, char *info ))
                              global_reference_table[TGR_get_status])
#define TGR_glob_flags   1
#define glob_flags  (*  (char *)  global_reference_table[TGR_glob_flags])
```

〔図8〕モジュール間のシンボル参照機構

〔図9〕モジュールコンフィギュレータのEXPORTシンボルの処理

ネルの動的サービスコールを呼び出すことでカーネル資源を生成します．

リスト1のTSKID_DEMOはユーザー指定の定義したタスクに割り付けられるタスクIDです．システムコンフィギュレータは，このようにID値を静的に割り当てます．割り当てた値(定数値)はC言語マクロヘッダファイル(kernel_id.h)に出力します．一方，ロードモジュールの場合，ロード時の機器の状態で割り当てるID値が異なってくるため，前述したような定数値にはできません．この理由により，ロードモジュールコンフィギュレータはID値を格納する変数領域を確保し，ローダ機能に出力するように実装しました．

また，機器で動作するローダ機能はロードモジュール活性時にロードモジュール構成情報を解釈し，自動ID採番付きのカーネルサービスコールを発行します．このとき採番されたID値は，前述のロードモジュールコンフィギュレータが確報した変数領域に格納するものとしました．しかし，アプケーションプログラムはこの内部的な差異を意識する必要はありません．このお陰で，開発初期段階にはベースモジュール機能として開発し，後でロードモジュール機能として切り出す，あるいはその反対の操作を行うことができます．

次にロードモジュールを構成するカーネル資源を，外部のロードモジュールに公開する場合，どのように開発者に記述させるかという課題が残ります．この課題に対して後述するシンボルのエクスポート機能を利用して，コンフィギュレータが確保した変数領域を公開することで可能なようにしました．

● ロードモジュール間のシンボル参照方法

ロードモジュールはすべてのシンボル参照関係を解決したリンク単位のオブジェクトモジュールです．機器ごとに異なるアドレスに配置される，他のロードモジュールの変数や関数を直接参照することはできません．ロードモジュール間で変数や関数を共有するしくみとして，モジュール内のシンボルエクスポート機能を実装しました．

開発者はモジュールコンフィギュレーションファイルに内部のシンボルを静的APIでエクスポート記述することで 外部のロードモジュールに公開することができます(**リスト2**)．

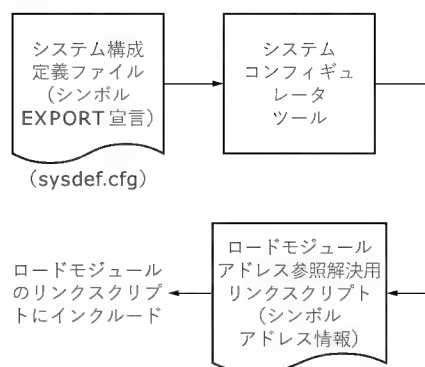
このシンボルのエクスポート情報は，ロードモジュールの付加情報として機器のローダ機能に出力されます．機器の内部では**図8**に示すように，システムで唯一のグローバル参照テーブルにシンボルのアドレス情報が設定されます．

他のロードモジュールがエクスポートされたシンボルを参照する際，このテーブルからアドレスを引いて間接参照することで可能になりますが，モジュールコンフィギュレータはエクスポートされたシンボルを参照するための手順をC言語のマクロ記述でグローバル参照ヘッダファイル(global_ref.h)に自動生成します．アプリケーションはこのヘッダファイルをインクルードすることにより，シームレスに他のロードモジュールのシンボル参照が可能なものとしました(**リスト3**，**図9**)．

ベースモジュールもロードモジュールからみれば，別のリンク単位であり，シンボルを直接参照することはできません．こ

〔図10〕システムコンフィギュレータのEXPORTシンボルの処理

〔図11〕ロードモジュールと保護ドメインの関係（その1）

〔図12〕ロードモジュールと保護ドメインの関係（その2）

れは，カーネルのサービスコールもC言語の関数として直接呼び出すことができないことを示しています．

これを解決する方法として，ベースモジュールもロードモジュールと同様に公開するシンボルをエクスポートする方法をとりました．ロードモジュール間のシンボル参照と異なる点は，ベースモジュール内のシンボルは機器ごとに配置アドレスが変わることはなく，シンボルに対応するアドレス情報が存在すれば，直接参照が可能となることです．ベースモジュールにはカーネルサービスコールを含む使用頻度の高い基幹機能が存在するため，間接参照によるオーバヘッドはなくしたいと考えました．ベースモジュールのエクスポート処理ではシンボルのアドレス情報をリンクスクリプト形式で出力し，ロードモジュールのリンク処理において，このリンクスクリプトをリンカに読ませることで，ロードモジュールのベースモジュールシンボルアドレスを解決させるようにしました（**図10**）．

● ロードモジュールと保護ドメインの関係

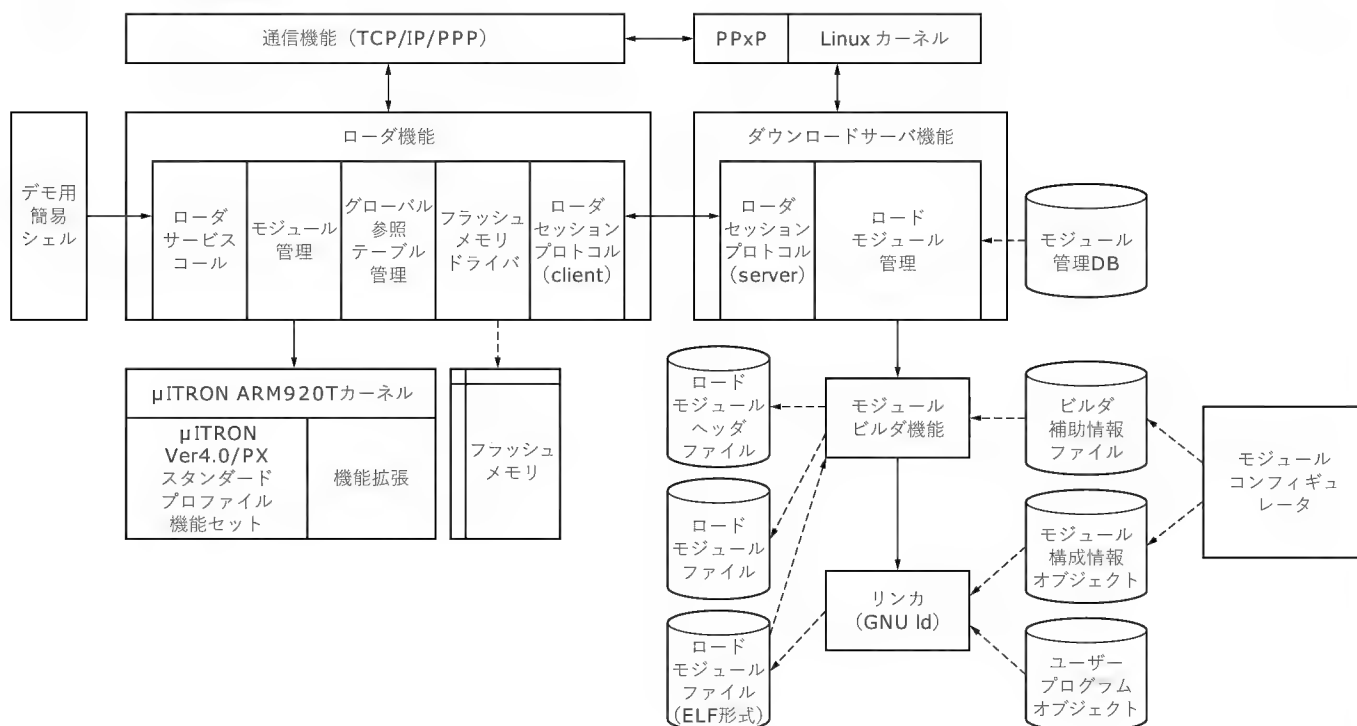
本プロダクトは，前述したようにμITRON保護機能拡張仕様のカーネル上にローダ機能を実装しています．保護機能拡張仕様では，「保護ドメイン」というオブジェクト（メモリ，カーネル資源）に対するアクセスの壁を任意に設定することが可能です．ロードモジュールと保護ドメインの関係をどのように考えるかという検討において，**図11**に示すようなロードモジュールと保護ドメインを一致させる方法が考えられます．このモデルではLinuxやWindowsの実行モジュール（プロセス）と本プロダクトのロードモジュールは近い概念となります．

しかし，TOPPERS/IDLが想定する比較的中規模なシステムにバージョンアップ機能をもったITRONが採用されるケースを考えた場合，**図12**に示すようにロードモジュールは「抽象的な機能単位」ではあっても，「保護される単位」ではないとしたほうが柔軟性があると考えました．また，開発者責任で保護ドメインが設定可能であれば，ロードモジュール単位の保護設定も可能となります．

以上の理由により，ロードモジュールの単位で保護ドメインの自動設定は行わず，開発者によるモジュール構成編集作業で保護ドメインを設定するようにしました．

● フラッシュメモリドライバについて

TOPPERS/IDLでは，ターゲット機器のプログラムを記憶するメモリとして，フラッシュメモリにプログラムを格納するシステムを想定して開発を行いました．初期のフラッシュメモリは，書き込みや消去操作中はチップの全領域がCPUから読めなくなりました．これは，システム動作を中断することなくダウンロードを実行するときに，問題となってしまいます．しかし，最近のフラッシュメモリではチップ内を複数のバンクで分割し，書き込みや消去操作中でも別バンクであれば，CPU

〔図13〕ソフトウェア構成図

から読み出し可能なチップが多くなってきています．TOPPERS/IDLではこのようなチップを使って，基幹機能とそれ以外の機能とでロードするバンク領域を分けるという方法を採用しています．これでフラッシュメモリ書き替え中にシステム動作が中断してしまうという問題を回避しています．

TOPPERS/IDLのソフトウェア構成

前節では詳細機能について説明しましたが，TOPPERS/IDLのソフトウェア全体の構成を説明します．

大別すると，

- Linuxサーバ上で動作するプログラム
- ターゲットシステム上で動作するプログラム

に分けられます．

サーバ上で動作するプログラムとしては，モジュールコンフィギュレータとダウンロードサーバ機能が含まれます．ダウンロードサーバ機能は，ターゲットとの通信を管理するセッションプロトコルと，ロードモジュールを管理するソフトウェアが含まれます．

一方，ターゲット上ではローダ機能，通信機能，シェル機能，μITRON4.0/PXカーネル機能が含まれます．ローダ機能には，ダウンロードされたモジュールを活性状態にするために実装されたローダサービスコール機能，モジュール管理機能，グローバル参照テーブル機能，フラッシュメモリドライバ機能と，Linuxサーバとの通信を管理するセッションプロトコルが含まれています．通信機能は，**図13**ではTCP/PPxPの例を示しています．無線や有線を使った通信でも，ユーザーが使用できる通信手段を使うことができます．

TOPPERS/IDLを活用したシミュレータ「Paratizer」

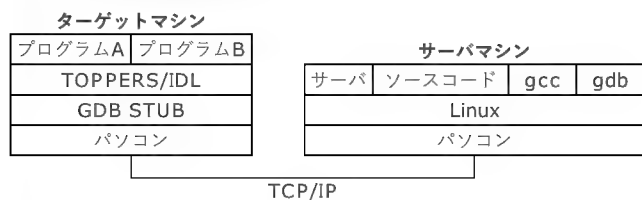
μITRONのスケジューリング規則に厳密にしたがってシミュレーションをするためには，Windowsなどの汎用OS上のシミュレータではなく，ハードウェアの上で直接ITRONを動作させるのがもっとも確実です．

そこで，パソコンの上でPentium用のITRON，TOPPERS/IDLを動作させ，デバッカやコンパイラを組み合わせたシミュレータ「Paratizer」を開発しました．Paratizerは大規模開発でも使用できるように，フリーで使えるGCCとGDBを採用し，PCだけで開発が行えるなど，開発コストを抑えられるようにしています．TOPPERS/IDLを使用しているので，ダイナミックローディングの特徴を活かしたデバッグが行えます．

システムは，**図14**のようにネットワークで接続したサーバマシンとターゲットマシンから構成されます．ターゲットマシンではPentium用のTOPPERS/IDLが動作しています．デバッグするプログラムはTOPPERS/IDLの上で動作します．ビルドサーバでは，Linux上でターゲットマシンと通信を行うサーバプログラムが動作します．ターゲットシステムからの要求は，このサーバプログラムによってハンドリングされます．

プログラムの開発方法は，次のようになります．

〔図14〕Paratizer構成図

〔図15〕従来の全体ダウンロードと部分ダウンロードの比較

1) まずビルドサーバ上にアプリケーションプログラムのソースコードを用意する
2) ソースコードをgccでコンパイル，リンクする
3) gdbを起動
4) gdbでターゲットマシンと接続
5) 2)で作成したプログラムをgdbでダウンロードして実行
6) gdbでデバッグする
7) 不具合が見つかったらソースコードを修正して再度コンパイルする
8) TOPPERS/IDLのシェル機能を使って修正されたプログラムのみをダウンロードする(全体をダウンロードする必要はない)
9) 6)～8)の工程を繰り返してプログラムを完成させる

いままでのISSやAPIシミュレータのようにプログラムのデバッグ，ソースコード修正，コンパイル/リンク，プログラム全体のダウンロードという開発サイクルではなく，デバッグしたいプログラムだけをダウンロード，デバッグできる点が大きく異なります(**図15**)．毎回プログラム全体のダウンロードをしなくてよく，しかも，ターゲットをリセットする必要もありません．デバッグしたい部分だけを何度でもダウンロードできます．デバッグしない部分のプログラムは，RUN状態にしておくことも休止状態にしておくこともできます．

PCの上でITRONを動作させるので，実行する命令はPentiumの命令です．そのためアセンブラでの動作確認は意味をもちません．しかし，アプリケーションプログラムの多くの部分はC言語で書かれているので，C言語で書かれた部分はこのシミュレータを使用してデバッグすることが可能です．変数の値を読んだり変更したり，構造体の中身を読んだりすることも可能です．このシミュレータを使ってアプリケーションプログラムの論理的な設計を行うことで，早期に開発着手できて，効率よくデバッグを行えます．

Paratizerのおもな特徴を以下に示します．

● シェル機能

ターゲットマシンでは，TOPPERS/IDLが動作していますが，ユーザーのデバッグ対象プログラムを読み込み，実行させるには，ターゲットマシンからトリガをかけ，サーバマシンからプログラムをダウンロードするための指示をする必要があります．トリガをかけるには，ターゲットマシンに組み込まれたシェル機能を使用して行います．この機能を使うには，シリアルケーブルでターゲットマシンと通信ソフトを実行するPCを接続し，通信ソフトからコマンドを入力します．入力されたコマンドは，ターゲットマシンで解釈され，実行されます．シェル機能として以下の機能が実装されています．

1) サーバマシンのダウンロード可能なプログラムのリスト表示
2) サーバマシンからのプログラムのダウンロード
3) ターゲットマシンの実行可能なプログラムのリスト表示
4) ターゲットマシンの実行可能なプログラムの実行
5) ターゲットマシンで実行中のプログラムの停止
6) ターゲットマシンの不要プログラムのアンロード

● ネットワーク機能とファイルシステム

通常のμITRON4.0仕様OSでは，ネットワークプロトコルやファイルシステムは実装されていません．しかしこの開発環境がベースとしているTOPPERS/IDLでは，ネットワークプロトコルとしてTCP/IPを標準で実装しています．ネットワークアプリケーションプログラムの開発者は，この機能をアプリケーションから呼び出し，使用できます．シミュレータでありながらデータの入出力が行えるので，通信相手と接続してプログラムの動作確認やデバッグをすることが可能になります．TOPPERS/IDLでは同様にファイルシステムを実装しています．Windows互換のFATファイルシステムとして使用できます．ターゲットマシンのハードディスクからのデータの読み出しや，書き込みなどの操作が行えます．アプリケーションプログラムでSDやCFといった小型メディアを使用するときには，ハードディスクを代替に使用してアプリケーションプログラムを作成しておくことで，早期にプログラム開発に着手できます．

● メモリ保護機能

μITRON4.0/PX仕様に準拠して実装されています．プログラムをドメインというサブシステムに分割し，各ドメインにアクセス許可ベクタを設定することが可能です．アクセス許可ベクタを設定することで，デバッグ対象のプログラムがアクセスしてよい領域といけない領域を明確に分離します．アクセスしてはいけない領域にアクセスした場合，カーネルが不正アクセスとして検出して例外処理ルーチンを起動します．例外処理ルーチンで不正処理を行ったプログラムの情報を取り出すことで，プログラム開発者は，迅速に，信頼性の高いプログラムを書くことができます．

〔図16〕メモリ保護機能をつかったデバッグ

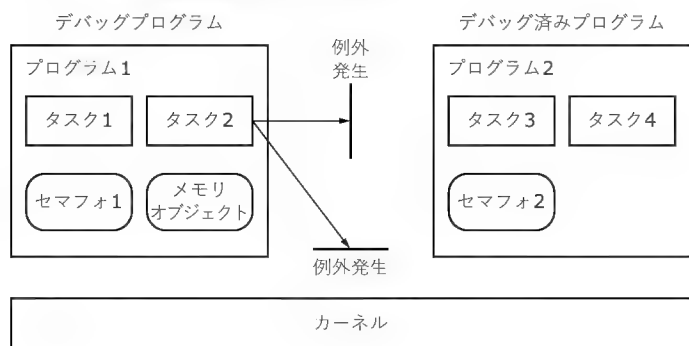



図16は，お互いに独立した二つのプログラムを開発している例です．お互いのプログラムは相互にアクセスされないこととします．プログラム1はデバッグ中，プログラム2はデバッグが済んだプログラムです．プログラム1をドメイン1に割り当て，プログラム2をドメイン2に割り当てます．ドメイン1の中のタスク2が，アクセスしてはいけないドメイン2のタスク3にアクセスしたり，カーネル領域にアクセスすると，例外が発生します．例外処理ルーチンでアクセスした不正なタスク2の情報を取り出し，なぜアクセスしないはずの領域にアクセスをしたかを確認することができます．

このようなメモリ保護機能を使わないと，デバッグ時は不正アクセスをしても偶然データが読めて，運用時にはデータが読めないといったことが発生します．大規模プログラムだと，こような不具合の発見には時間と労力が必要になりますが，メモリ保護機能を有効に活用することでプログラムのテスト段階でバグとして検出でき，デバッグ効率が向上します．

● I/Oシミュレーション機能

プログラムが大規模化するにしたがってプログラム単位でミドルウェアとして購入し，利用することが多くなっています．ミドルウェアを購入するとしても，同様の機能をもつミドルウェアが複数のベンダから提供されていて選定に時間がかかっていたり，購入の手続きに時間がかかるなどで開発着手時にはミドルウェアが入手できない場合があります．また，調査目的でミドルウェアを使用したいということもあります．ここで説明するI/Oシミュレーション機能を使うことで，使用したいミドルウェアが存在しない場合でも，あたかも実装されているかのようにプログラムの作成とデバッグを行えます．

図17は，ターゲットマシンには存在しないIPv6のミドルウェアをI/Oシミュレーション機能を使ってシミュレーションする例です．ターゲットマシンではIPv6を使用するアプリケーションClient Applicationを作成します．このプログラムで使用されるIPv6のソケット関数は，まずRPC STUB関数を経由してサーバマシンに転送されます．サーバマシンでは，RPC STUB関数でIPv6 Socketに接続します．ここからはLinuxのIPv6のデータとしてI/Oシミュレーション用PCに転送されます．ここでは受け取ったデータでデバイスを動作させて，その結果を再びサーバマシンに転送します．サーバマシンのRPC STUB関数で再びIPv4のソケットに接続され，ターゲットマシンにデータが転送されます．なお，I/Oシミュレーション用PCは特別に設けなくても，サーバマシンで共用することも可能です．

I/Oシミュレーションを行うには，RPCサーバSTUB関数を作成しておきます．**リスト4**はClient Applicationで使用する

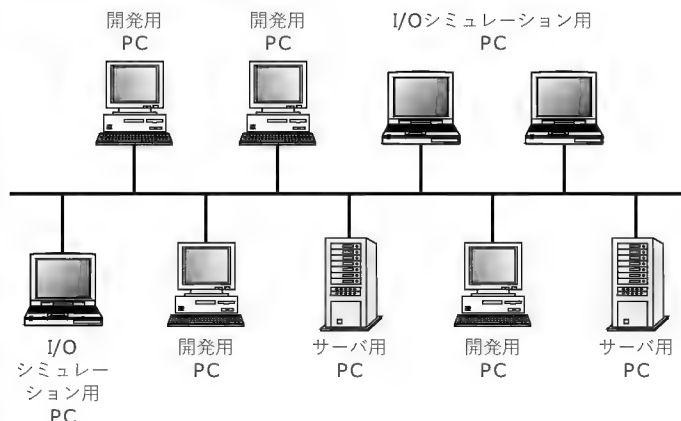
〔リスト4〕RPCサーバSTUB関数例

```
int * iosim_socket_1 ( struct socket_arg *argp,
                       struct svc_req *rqstp)
{
  static int  s;
  int         family, so_type, proto;
  switch ( argp->family )
    {
    case IOS_IP4:
      family = AF_INET;
      break;
    case IOS_IP6:
      family = AF_INET6;
      break;
  }
  switch ( argp->so_type )
    {
    case IOS_STREAM:
      so_type = SOCK_STREAM;
      break;
    case IOS_DGRAM:
      so_type = SOCK_DGRAM;
      break;
    }
  proto = 0;
  printf ("server socket ()  call family[%d] type[%d]¥n",
         family, so_type ) ;
  s = socket (family, so_type, proto) ; /* IPv6ソケット作成 */
  printf ("server socket ()  call end so [%d]¥n",s ) ;
  return  (&s) ;
}
```

〔図17〕I/Oシミュレーション動作概要

〔リスト5〕RPC　I/F定義ファイルの例

```
program IOSIMPROG {
    version IOSIMVERSION {
        int  IOSIM_SOCKET (socket_arg)  = 1 ;
        int  IOSIM_SENDTO (sendto_arg)  = 2 ;
        recvfrom_res  IOSIM_RECVFROM (recvfrom_arg)  = 3 ;
        int  IOSIM_CLOSE ( int so )  = 4 ; // 関数番号定義
    } = 1 ; // バージョン番号定義
} = 0x20000000 ; // プログラム番号
                 // (ユーザ定義用は0x20000000～0x3fffffffを使用)
```

〔図19〕大規模開発での使用例

関数の一つsocketを作成する例です．他の関数(sendto，recvfromとclose)も同様に作成しておきます．

次にRPC言語でRPC I/F定義ファイル(**リスト5**)を記述します．プログラム番号とバージョン番号および関数番号を記述します．関数番号はClient Applicationで使用する関数socket，sendto，recvfrom，closeそれぞれを定義します．定義ファイルは拡張子.xで作成し，RPCコンパイラ(rpcgen)にかけます．

定義ファイルを作成したら，RPCコンパイラで以下のようにコンパイルします．

```
%rpcgen iosim.x
```

コンパイルが終了するとRPCクライアント側(ターゲットマシン側)のSTUBルーチンioxim_clnt.c，RPCサーバ側(サーバマシン側)のRPCサーバmain()関数iosim_svr.cと，クライアントサーバ共通ルーチンiosim_xdr.cとヘッダファイルiosim.hが出力されます．RPCクライアント側STUBルーチンと共通ルーチンは，ITRON側アプリケーションプログラムと共にコンパイル・リンクを行います．RPCサーバmain()関数と共通ルーチンは，RPCサーバSTUB関数とともにコンパイル/リンクしてサーバマシンで実行させます．さらにIPv6のパケットを受け取るプログラムをServer Applicationとして実行させておきます．これでI/Oシミュレーションに必要な準備はすべて終了です．Client Applicationを実行させると，パケットはサーバマシンを経由してIPv6のパケットを受け取るServer Applicationに転送されます．Server Applicationでの実行結果はClient Applicationに転送されてきます．これでシミュレータには存在しないミドルウェアを用いたプログラムの開発が可能なことが理解いただけたと思います．

〔図18〕スタンドアロンモードでの実行

Paratizer
VMWare
Linux
PC

● Paratizerの運用方法

Paratizerの特徴の一つに，複数のPCとネットワークを使用してデバッグを行うことがありますが，ここでは，各種の運用形態を紹介します．

▶**スタンドアロンモード**

スタンドアロンモード(**図18**)は，1台のPCでデバッグを行えるようにしたもので，VMWareを併用することで実現します．VMWareの，1台のPCで複数のOSを同時に動作させる機能でLinuxとParatizerを同時に実行させます．パソコン1台でデバッグができるので，小規模なプログラム開発に向いています．

▶**LANモード**

大規模なプログラムの開発では，多くのプログラマが同時に開発を行います(**図19**)．このような状況では，サーバマシンとターゲットマシンを複数使用したり，サーバマシンと開発用PCを別PCにすることで使い勝手を向上させることができます．I/Oシミュレーション用PCも，使用するI/Oごとに準備したほうがシミュレーション時の負荷を軽減できます．

ユーザーの開発スタイルに合わせて適宜構成を変更できるのが，LANモードのメリットです．

● Paratizerの今後の展開

TOPPERS/JSPはスタンダードプロファイルに対応したリアルタイムカーネルです．ParatizerはTOPPERS/JSPをベースに開発されているので，シミュレーションできるAPIは，TOPPERS/JSPがサポートしているAPIになります．一方，ユーザープログラムではμITRON4.0仕様の拡張機能を使用する場合もあります．ユーザープログラムで使用されるAPIはParatizerでも利用できることが望ましいので，順次サポートAPIを増やしていく予定です．また，Linuxサーバマシンで実行させるサーバプログラムをWindows上で実行したいという要求も非常に高いので，Windowsに対応したサーバプログラムの開発も予定しています．

かわい・たかお　(株)エーアイコーポレーション

海外・国内イベント/セミナー情報

INFORMATION

海外イベント

11/3-7 **2003 Fuel Cell Seminar**
Fontainebleau Hilton, Miami Beach, FL, USA
Fuel Cell Seminar Headquarters
http://www.fuelcellseminar.com/

11/4-7 **2003 Asia-Pacific Microwave Conference**
Sheraton Walker Hill Hotel, Seoul, Korea
GENICOM Convention Service, Co., Ltd.
http://www.apmc2003.org/

11/15-19 **20th International Electric Vehicle Symposium and Exposition**
Long Beach Convention Center, Long Beach, CA, USA
Electric Drive Transportation Association
http://www.evs20.org/

11/15-20 **COMDEX Fall 2003**
Las Vegas Convention Center, Las Vegas, NV, USA
COMDEX Registration and Housing
http://www.comdex.com/lasvegas2003/

11/16-20 **2003 ITS WORLD CONGRESS**
Spanish capital, Madrid, Spain
ITS Japan
http://www.madrid2003.itscongress.org/

12/7-10 **2003 IEEE International Electron Devices Meeting**
Hilton Washington and Tower, Washington, D.C., USA
Institute of Electrical and Electronics Engineers, Inc.
http://www.his.com/~iedm/

12/9-11 **Bluetooth Americas 2003**
San Jose Convention Center, San Jose, CA, USA
Informa UK Limited
http://www.ibctelecoms.com/bluetoothamericas/

国内イベント

11/5-7 **計測展 2003 TOKYO**
東京国際展示場(東京ビッグサイト，東京都江東区)
(社)日本電気計測器工業会
http://expo.nikkeibp.co.jp/jemima/

11/11-13 **Mobile & Wireless World/Tokyo 2003**
東京国際フォーラム(東京都千代田区)
(株)IDGジャパン Mobile & Wireless World 事務局
http://www.idg.co.jp/expo/mww/

11/12-14 **Embedded Technology 2003/組込み総合技術展**
パシフィコ横浜(神奈川県横浜市)
(株)アイシーエス企画 Embedded Technology 2003 運営事務局
http://www.jasa.or.jp/et/

11/19-22 **2003国際ロボット展**
東京国際展示場(東京ビッグサイト，東京都江東区)
(社)日本ロボット工業会
http://www.nikkan.co.jp/eve/03robot/index.html

11/28 **第6回 五大都市FPGAカンファレンス2003 ーなにわFPGAカンファレンス**
梅田センタービル(大阪府大阪市)
セイコーインスツルメンツ(株)内 FPGAコンソーシアム事務局
http://www.sii.co.jp/eda/fpga/

12/2-5 **Internet Week 2003**
パシフィコ横浜(神奈川県横浜市)
Internet Week 2003 登録事務局
http://internetweek.jp/

12/3-5 **'03国際画像機器展**
パシフィコ横浜(神奈川県横浜市)
精機通信社
http://www.seiki-tsushin.com/ite/

開催日，イベント名，開催地，問い合わせ先の順

セミナー情報

USBデバイス開発・設計手法
開催日時　：10月28日(火)～10月29日(水)
開催場所　：オームビル(東京都千代田区)
受講料　　：68,500円(1口で1社3名まで受講可)
問い合わせ先：(株)トリケップス，☎(03)3294-2547，FAX(03)3293-5831
http://www.catnet.ne.jp/triceps/sem/c031028a.htm

ICタグ そのしくみとアプリケーション
開催日時　：10月31日(金)
開催場所　：SRCセミナールーム(東京都高田馬場)
受講料　　：48,000円
問い合わせ先：(株)ソフト・リサーチ・センター，☎(03)5272-6071
http://www.src-j.com/seminar_no/23/23_207.htm

LANのしくみとEthernet技術解説
開催日時　：11月4日(火)～11月5日(水)
開催場所　：SRCセミナールーム(東京都高田馬場)
受講料　　：59,000円
問い合わせ先：(株)ソフト・リサーチ・センター，☎(03)5272-6071
http://www.src-j.com/teiki_no/TOMARU/tomaru_2.htm

無線データ通信の基礎と2.4GHz帯無線LAN
開催日時　：11月7日(金)
開催場所　：CQ出版セミナールーム
受講料　　：13,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎(03)5395-2125，FAX(03)5395-1255

入門Linuxデバイスドライバ開発技法
開催日時　：11月10日(月)～11月11日(火)
開催場所　：オームビル(東京都千代田区)
受講料　　：62,500円(1口で1社3名まで受講可)
問い合わせ先：(株)トリケップス，☎(03)3294-2547，FAX(03)3293-5831
http://www.catnet.ne.jp/triceps/sem/c031110a1.htm

UMLによるオブジェクト指向システム分析/設計コース
開催日時　：11月11日(火)～11月12日(水)
開催場所　：SRCセミナールーム(東京都高田馬場)
受講料　　：76,000円
問い合わせ先：(株)ソフト・リサーチ・センター，☎(03)5272-6071
http://www.src-j.com/teiki_no/UML/uml_03.htm

XML基礎
開催日時　：11月12日(水)
開催場所　：NRI大手町ラーニングセンター(東京都千代田区)
受講料　　：42,000円
問い合わせ先：(株)エイチアイICP事業部，☎(03)3719-8155，FAX(03)3793-5109
http://icp.hicorp.co.jp/seminar/nrixml/xml_kiso.asp

Hi-Speed USB基礎コース
開催日時　：11月14日(金)
開催場所　：半導体トレーニングセンター 飯田橋会場(東京都新宿区)
受講料　　：10,000円
問い合わせ先：(株)ルネサステクノロジ 半導体トレーニングセンター，☎(03)3266-9344
http://www.renesas.com/jpn/support/seminar/

ATA(IDE)/ATAPI技術解説
開催日時　：11月18日(火)～11月19日(水)
開催場所　：オームビル(東京都千代田区)
受講料　　：68,500円(1口で1社3名まで受講可)
問い合わせ先：(株)トリケップス，☎(03)3294-2547，FAX(03)3293-5831
http://www.catnet.ne.jp/triceps/sem/c031118a.htm

Win32APIネットワークプログラミング
開催日時　：11月25日(火)～11月26日(水)
開催場所　：ディーアイエステクノサービス研修室(東京都文京区)
受講料　　：98,000円
問い合わせ先：(株)エイチアイICP事業部，☎(03)3719-8155，FAX(03)3793-5109
http://icp.hicorp.co.jp/seminar/c-vc/vc_net.asp

画像処理のためのインターフェース開発技術
開催日時　：11月26日(水)～11月28日(金)
開催場所　：高度ポリテクセンター(千葉県千葉市)
受講料　　：35,000円
問い合わせ先：雇用・能力開発機構 高度ポリテクセンター事業課，☎(043)296-2582
http://www.apc.ehdo.go.jp/

SH-Linuxマイコン入門
開催日時　：11月27日(木)
開催場所　：CQ出版セミナールーム
受講料　　：13,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎(03)5395-2125，FAX(03)5395-1255

Webアプリケーションのセキュリティ対策
開催日時　：11月28日(金)
開催場所　：NRI大手町ラーニングセンター(東京都千代田区)
受講料　　：47,250円
問い合わせ先：(株)エイチアイICP事業部，☎(03)3719-8155，FAX(03)3793-5109
http://icp.hicorp.co.jp/seminar/nrisecu/secu_web.asp

ディジタル信号処理入門
開催日時　：11月29日(土)
開催場所　：CQ出版セミナールーム
受講料　　：13,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎(03)5395-2125，FAX(03)5395-1255

日程はすべて予定です．問い合わせ先にご確認のうえ，お出かけください．

Engineering Life in

ユーザーインターフェースのスペシャリスト（第二部）

■今回のゲストのプロフィール
ブラッド・ホックバーグ（Brad Hochberg）：ニューヨーク州ロングアイランド出身．スタンフォード大学にて情報工学科を卒業後，オラクルを経てアップルコンピュータに入社．アップルでは，Coplandなど MacOSの主要プロジェクトにエンジニアとして参加．その後ユーザーインターフェースのスペシャリストとしてVTELのスタートアップ，OnScreen24に行く．現在は，パーソナルビデオレコーダで著名なTiVoでユーザーインターフェースデザイナーとして活躍中．趣味はミュージカルや演劇の鑑賞，料理．最近はサイクリングとヨガに熱中している．

前回まで：もともとエンジニアをめざさず環境都市学を専攻していた話から，ソフトウェアの仕事についた話をうかがった．氏はアップルコンピュータでは，OS 8や9の設計に携わった人物でもある．アップル時代での「下積み」から話を続けてもらう．

☆ 念願のユーザーインターフェースの仕事に就く

トニー アップルでは，めざしていたユーザーインターフェースの仕事をされたのですよね？　ファインダ，コントロールパネルなどをやっていたわけですか？

ブラッド 一人何役もやっていましたが，正式なユーザーインターフェースグループの一員ではありませんでした．個人的には，かなりハードな日々が続いていて，自分のめざしているユーザーインターフェースのエンジニアとして早く専念したかったものです．そこでMacOS 8プロジェクトの終わりで区切り良い時期に配置換えを願い出たのですが，まだまだ経験が足りないといわれ，結局，かなりもめた後に却下されました．個人的には十分下積みをしたと思っていたので，かなり不服に感じ，アップルでは自分のやりたいことが実現できないのだと感じてきました．

また，同じころにお世話になった技術の取りまとめ役の方が辞めてしまいました．彼女は，スタートアップに行ってCTO（Chief Technology Officer）になられたので，私もそこに行くことにしました．ビデオ会議システムを作っていたOnScreen24です．ここで初めて，念願のユーザーインターフェースのデザイナとして専念することができるようになりました．

トニー スタートアップの経験は，ここがはじめてですね？

ブラッド そうですね，ビデオでのメールシステムを作っていました．結局，親会社のVTELからの追加投資がうまく得られず，解散しました．2000年秋のことです．幸いまだそのころはインターネットバブルが弾ける前だったので比較的に簡単に仕事が見つかり，現在のTiVoに入社しました．TiVoでは引き続き，ユーザーインターフェースのスペシャリストとして仕事をしています．

☆ ユーザーインターフェースの仕事とは？

トニー それでは実際に，ユーザーインターフェースの仕事について語っていただけますか？

ブラッド まず，ユーザーインターフェースは会社の方針や，製品によって，取り組みがだいぶ違ってくると思います．たとえば，冷たい水と熱いお湯が出るオフィス用の給水器ですが，熱湯が出るスイッチやコックにうっかり触れ，熱湯で火傷をすることがないようにします．たとえば，解除ボタンを押してから熱湯が出るボタンを押すとかですね．ものによっては，操作法が直感的ではなく，使い方に苦しむ製品もあります．

ユーザーインターフェースには，その会社の方針とか製品のタイプが出ていると思います．良いユーザーインターフェースには，やはりデザインとエンジニアリングの時間とコストがかかるからですね．一方，広くコンシューマに使われる製品だともう少しユーザーの使ったときの体験を考えなければヒット製品になりません．たとえばアップルのMacユーザーは熱狂的で，ファンが基盤として確実に存在すると思うのですが，それと同じレベルにするにはユーザーの体験が素晴らしいものでなければなりません．

トニー 最近気に入った製品などはありますか？

ブラッド 個人的には，アップルのiPodがなかなかスマートな製品だと思います．他のMP3プレーヤをいくつか使ってみたのですが，アップルらしいスマートさがあります．こういうのはアップルがやはり上手だと思います．

トニー TiVoもユーザーにすごく支持されている製品ですよね．熱心なユーザーにより，TiVoのハックサイト[注1]が運営されており，さまざまなハックのやり方の紹介がされていたり，実際にユーザーがさまざまな改造をしているとか……．

ブラッド おっしゃるとおりで，TiVoもMacユーザーのような熱狂的なファンベースを作ることを優先課題としています．会社によっては，ユーザーインターフェースをエンジニアリングの一部ととらえる会社もあれば，マーケティングに近いグループと考える場合もあります．TiVoの場合は中間です．

実際の仕事のプロセスですが，まずは，製品開発の段階でマーケティングとエンジニアリングの要望を揃えて調整することからはじめます．TiVoには実際製品を製造するメーカーとのパートナーシップもあるので，その要望も取り入れる必要があります．エンジニアリングやパートナーの要望というのは，技術的に影響するものが多いです．たとえば，リモコンにボタンを付けたほうが利便性がアップする場合もありますが，部品コストに直接的な影響があるし，すでに出荷されているユーザーへのアップデートもコスト面での影響があります．そういう場合は，オンスクリーンのメニューに取り込んでしまうというスタイルになります．

トニー エンジニアリング，マーケティングそしてサードパーティのメーカーとの綱引きのようなのですが，妥協が多いのではないでしょうか？

注1：TiVo Community Forumと呼ばれるサイト．TiVoは基本的に80GバイトのHDDをもったLinuxマシンなので，ハックや改造が非常に人気を呼んでいる．メールのチェックやWebブラウズの方法などが紹介されている．

Silicon Valley

H. Tony Chin

対談編

ブラッド 何らかの妥協はもちろんあります．たとえば最近の例で，ある日本のメーカーと共同開発した製品があるのですが，われわれの意向としてはリモコンをTiVo特有のシンプルで楽しさを訴えるようなスタイルにしたかったのですが，このメーカーはAV製品のリモコンを統一させたデザインにしているので，彼らの統一デザインに落ち着きました．堅い雰囲気になり，個人的にはあまりよくないと思いましたが(苦笑)．

トニー 良いユーザーインターフェースの決め手となるものはあると思いますか？

ブラッド そうですね，もっとも大事なのは顧客……ユーザーを知ることだと思います．これによってユーザーの体験をよりいっそう素晴らしいものにできるからです．ユーザーを知るには，さまざまな方法で調べることが可能です．たとえば，マーケティングの力を借りてアンケートをしたり，フォーカスグループを行ったりします．アンケートやフォーカスグループでは，質問の仕方が答の方向を決めてしまうことがあるので，質問の設定やヒアリングの進め方についての「設計」についても，ユーザーインターフェースのスペシャリストとして関与することがあります．

トニー それは，どういうことですか？

ブラッド ユーザーのやりたいことと言っていることに隔たりがある場合があるからです．ですから，アンケートの質問の内容をマーケティングの人達と考える部分にも参加する場合があります．これは，エンジニアリングのスキルというよりは，心理学的な要素，そして観察力の要素が必要ですね．

次に，実際に答えが揃ってきたら，さらにくわしく聞くためにフォーカスグループを行ったり，ヒアリングを行ったり，実際のモックを見せたりしていきます．プロトタイプの段階になるとユーザーテスティングのラボで実際の製品に触れてもらいます．ここでも，どのような人をテストの対象にするか？またどのような内容のテストにするか？これらが非常に結果を左右します．

トニー つまりテストの設計の仕方によっては，ズルができるのですよね．良い結果にしてしまうとか…… ちょっと話がずれますが，私が以前勤めていたGeneral Magicでたまたま自分のオフィスがユーザーテスティングのラボの近くだったのですが，警察署の取調べ室みたいでした(笑)．マジックミラーになっていて，何台かのビデオカメラがユーザーの状況を録画しているのですよ．かなりの本数のビデオテープがありました．

ブラッド ユーザーを連れてきてテスティングをする部屋ですね．われわれの使う道具の一つです．しかし，テストの設計の仕方や仮説の立て方によって，結果は思ったとおりにならないものです．

例を挙げると，最近，衛星テレビのDirecTVと共同開発した製品をテストしました．箱から開けて初期設定をする部分がおもなテストでした．DirecTVは100チャネル以上あり，スポーツの中継が充実していることで有名です．最近はHDTVも始まったので，それのチューナ込みの機種をテストしました．初期のターゲットとなる顧客は，かなりテレビが好きな若い男性ですよね……またシリコンバレーなので，かなり技術的なことに慣れている人達……こういう設定でテストを考えたのですが，実際の結果は皆さんかなり苦労していました．HDTVのチャネル設定の部分とかアンテナの調整とかが難しかったようです．こういうデータを元にまたデザインの再検討を行ったり，場合によっては，ユーザーマニュアルの強化などの具体的な解決策に結び付けていきます．

ブラッド・ホックバーグ氏

☆ ユーザーの体験を素晴らしいものにする

トニー 実際は，かなり地味なブラッシュアップ的なプロセスなのですね．でも，大学で学んだ環境都市学のプロセスに似ているとおっしゃっていたことが何か理解できるような気がします．

ところで，まったく違ったスタイルのマンマシンインターフェースを考えたいとかありますか？

ブラッド 今のところはないですね(笑)．スタートアップなので，早く黒字転換してほしいです．われわれの製品がやはりリビングルームでの娯楽マシンとしての位置付けなので，テレビを中心とした娯楽をいかに楽しくするか？というのが私の課題です．テレビが大好きで，そして楽しく使えるマシンをめざしています．楽しくするために，キュートなマスコットがアニメーションとしてメニューで動くとかさまざまな工夫をするわけです．これは，専属のグラフィックアーティストが取り組んでいます．また，楽しくするにはユーザーのやりたいことをドンドン取り込むことです．最近デジカメで取った写真を取り込んで音楽と一緒にスライドショーみたいに見せる機能を取り込んでいますが，これもじつは，ハックサイトで提案されていたものをヒントにしています．ユーザーの体験をよりいっそう素晴らしいものにするのが，私のめざしているものです．

対談を終えて

ブラッド氏はジムでお会いしたエンジニアの一人だ．シリコンバレーでは稀なコンシューマー系の家電の仕事についているので，なかなか興味深い話ばかりだった．ドンドン多くを語ってくれる優秀なゲストだった．アップルコンピュータでの裏舞台話もたくさん出たが，少し過激な内容もあったので，非常に残念だが割愛した．ブラッド氏は，エンジニアリング以外にもMultiple Sclerosis(多発性硬化病)の寄付金集めの100マイルサイクリングツアーの会社のリーダーとしても活躍している．

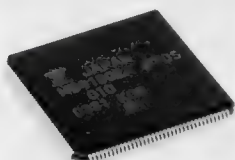
トニー・チン htchin@attglobal.net WinHawk Consulting

HARD WARE

●32ビットRISCマイコン

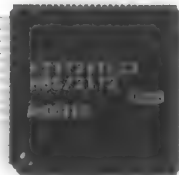
MB91302A-010

- リアルタイムOS「μ ITRON3.0」を内部メモリに搭載し，リアルタイムOSの購入費用を削減．
- リアルタイムOSをカスタマイズして導入する期間を省略し，アプリケーション開発に着手できるため，機器の開発期間を短縮．
- マイコン内部でタスクの切り替え処理を実行することが可能となり，高速で安定したリアルタイム処理を実現．
- プロセステクノロジは，CMOS 0.25μmプロセス．
- 動作電圧は，3.0～3.6V（標準：3.3V）．
- ROM，RAMおよびキャッシュメモリをそれぞれ4Kバイトずつ搭載．

■ 富士通（株）
サンプル価格：¥1,500
TEL ： 042-532-1397
E-mail ： edevice@fujitsu.com

●16ビット1チップマイコン

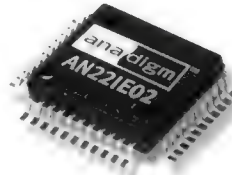
H8S/2172F

- USB 2.0に対応したモジュールを内蔵．コントロール，バルク，インタラプトの転送モードをサポートし，最大480Mbpsの転送速度を実現しているため，大容量のマルチメディアデータなどの高速転送が可能．
- メモリースティックPROに対応したインターフェースの内蔵に加え，著作権保護技術のマジックゲートを搭載．
- 16ビットマイコン「H8S/2000」CPUコアを搭載し，33MHz動作を実現．
- 内部データバス幅を16ビットから32ビットに拡張し，データ転送速度を向上．
- 256Kバイトのフラッシュメモリを内蔵しているため，各種のデバイスドライバの格納やプログラムの書き換えが容易．

■（株）ルネサス テクノロジ
サンプル価格：¥1,700
TEL ： 03-5201-5276

●プログラマブルアナログアレイ

形AN221E02

- パソコンとテストボードを用いてアナログICを機能仕様レベルで設計し，その場でチップに書き込んで使用できるため，設計や試作期間を大幅に短縮．
- スイッチトキャパシタ法により，従来のアナログ設計の温度変化などの制約を解消し，温度変化に強いアナログICを実現可能．
- OPアンプ4個，コンパレータ2個と内部回路規模を従来品の半分に小規模化することで，コストを3割弱低減．
- 入力/出力数はともに2本．マルチプレクス機能を使えば5入力まで拡張可能．
- 設計ツール上で機能ブロックを選択，配線，仕様を設定するだけでアナログICの設計が完了．

■ オムロン（株）
価格：¥600～（1,000個時）
TEL ： 075-344-7074

●ディスプレイコントローラLSI

MB86275

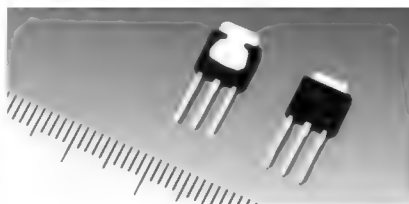
- SiP技術により，64MビットFCRAMとロジックチップを1パッケージ化．
- 基板上にメモリとロジックチップを個別に実装した場合よりも，画像処理部の実装面積を約半分に縮小できるため，携帯型AV機器の小型化を実現できる．
- 各部へのクロック入力をコントロールし，無駄な電力消費を回避するパワーマネージメント機能を内蔵することで，画像表示時でも110mWの低消費電力を実現．
- ITU-RBT601/656に準拠したビデオ映像を取り込めるビデオキャプチャ機能を搭載．
- 取り込んだビデオ画像は，拡大，縮小表示することが可能．
- 最大4,096×4,096ドットの解像度のビデオ画像を取り込む機能を搭載しているため，ディジタルカメラなどの高解像度の映像の取り込みが可能．
- QVGAからXGAまでの解像度に対応しているため，さまざまなディスプレイ表示に対応可能．

■ 富士通（株）
サンプル価格：¥9,000
TEL ： 03-5322-3325
E-mail ： edevice@fujitsu.com

●スイッチングデバイス

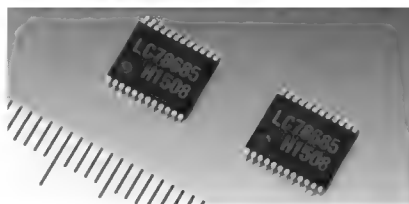
GTBTシリーズ

- 600V耐圧のMOSFETに対して2桁，バイポーラトランジスタ/IGBTに対して1桁低い，最小ON抵抗を実現．
- 電圧600V/定格電流2Aの場合，実装面積を85%低減可能．
- 従来品と比較して1/2の高速スイッチングを実現しているため，低損失電力変換，高周波化が可能．
- ON電流が負の温度係数を有しているため，従来品と比較して5倍の安全動作領域を実現．
- 従来品と比較して30倍の電流利得を実現し，低電力駆動が可能．
- 従来品と比較して6倍の電流密度を実現しているため，小型パッケージへの搭載が可能．

■ 三洋電機（株）
サンプル価格：¥20～¥200
TEL：0276-61-8055　FAX：0276-61-8854

●MP3/WMAデコーダLSI

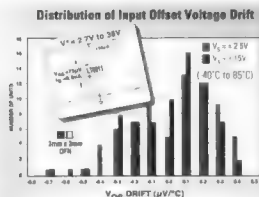
LC78685V

- CD-DSPおよびCD-MP3の基本構成に，本LSIを付加することでMP3/WMAマルチデコード対応が可能．
- SSOP24（8.0×5.6mm）と小型パッケージのため，省スペースでの実装が可能．
- CD/CD-MP3/CD-MP3-WMAの3種類のラインナップで用途にあわせた低コスト化を実現可能．
- 低消費電力のため，ポータブル機器の電池寿命を大幅に伸ばすことが可能．
- WMAデコーダ部は，WMAデコード，WMAヘッダ情報読み出し，フレームエラー自動リカバリなどの機能をもつ．
- オーディオ信号出力はLRCK，BCK，DATAのシリアル出力で，フォーマットはI^2S．

■ 三洋電機（株）
サンプル価格：¥500
TEL ： 0276-61-9600　FAX：0276-61-8849

●レールツーレールOPアンプ

LT6011

- 150μA以下の消費電流で動作し，75μV以下の入力オフセット電圧，0.9nA以下の入力バイアス電流，14nV/√Hzの低ノイズ電圧密度を達成．
- 最小120dBの開ループ利得によって優れた利得直線性を確保し，115dB以上のCMRRによって同相信号による誤差を低減．
- 0.8μV/℃の低い入力オフセット電圧ドリフトと0.4μV/月の長期安定度により，温度と時間の全範囲で高精度を保証．
- 入力オフセット電圧，入力バイアス電流，同相除去比，電源除去比などのすべてのパラメータに関して，チャネル間のアンプマッチングに優れる．

■ リニアテクノロジー（株）
参考価格：¥235～（1,000個時）
TEL：03-5226-7291　FAX：03-5226-0268

●TFT液晶ドライバ

HD66784

- 1チップでQCIFサイズ（176×224ピクセル）のメイン画面および176×96ピクセルのサブ画面を同時に駆動可能な，26万色表示対応のアモルファスTFT液晶ドライバ．
- 1チップでメインとサブ画面を同時に駆動できることから，部品点数の削減が可能．
- 26万色表示に対応した，約126Kバイトの表示用RAMと表示制御用のコントローラを内蔵し，メインおよびサブ画面とも高品質な画質を実現．
- 8色表示機能，スタンバイモードの低消費電力機能により，機器の低消費電力化を図ることができる．
- 待ち受け表示時のサブ液晶パネルの消費電力を，STN液晶パネルとほぼ同等の0.8mWまで低減可能．

■（株）ルネサス テクノロジ
サンプル価格：¥2,200
TEL：03-5201-5226

●SRAM

superSRAMシリーズ

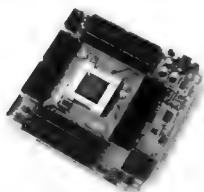
- SRAMセルとDRAMキャパシタを融合した新型メモリセルの開発により，大幅にチップサイズを低減でき，16Mビット低消費電力SRAMとしては小型のチップサイズ約32mm^{2}を実現．
- メモリセルに蓄えられた情報は自動的に維持されるため，リフレッシュは不要．
- データ保持電流は1μAで，バッテリ駆動の携帯機器の低消費電力を実現できる．
- メモリセルの記憶ノードにDRAMセルで実績のあるスタックトキャパシタを適用しているため，α線や中性子線によって引き起こされるソフトエラーが起こりにくい．

■（株）ルネサス テクノロジ
サンプル価格：¥1,800
TEL：072-784-7333

●FPGA評価ボード

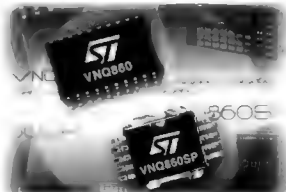
XSP-013

- ザイリンクス社の「Spartan-ⅡE」シリーズを実装した評価用ボード．
- 30万ゲートの「Spartan-ⅡE：XC2S300E」は，ユーザーI/Oが豊富なため，各種実験や評価に適する．
- オンボードのコンフィグレーションROMから，電源投入時にFPGAをコンフィグレーションさせることが可能．
- FPGAで必要となる「＋1.8V」と「＋3.3V」の電源はオンボードのレギュレータが生成するため，別途，電源装置は不要．
- 「Spartan-ⅡE：XC2S300E」よりも大きなゲート数にも対応可能．

■（有）ヒューマンデータ
価格：¥75,000～¥195,000
TEL：072-620-2002　FAX：072-620-2003
E-mail：Sales2@hdl.co.jp
URL：http://www.hdl.co.jp/

●4チャネルハイサイドドライバIC

VNQ860 VNQ860SP

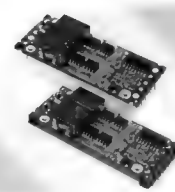
- 同社のVIPower M0-3プロセス技術を使用して製造されたモノシリックデバイスで，片側がGND接続されているものであればどのような負荷でも駆動できる．
- 過電流制限，過熱保護，自動リスタートなどの過負荷保護機能を装備．
- GND接続が切断されると自動的にOFFになり，電圧が過小または過大になると自動的にシャットダウンする．
- ショートに対する保護機能を装備．
- CMOS対応の入出力を備えており，スタンバイ電流は低く抑えられている．
- 0.25A以下の電流域で負荷駆動．

■STマイクロエレクトロニクス（株）
サンプル価格：VNQ860 ¥300（100個時）
VNQ860SP ¥360（100個時）
TEL：03-5783-8260　FAX：03-5783-8216

●DC-DCコンバータ

LENシリーズ

- 1/8ブリック型，高効率，単出力非絶縁型のDC-DCコンバータ．
- 鉛フリーに対応したオープンフレーム形状で，表面実装モデルとピン実装モデルの2種類を用意．
- 分散型電源システム構築に適し，出力電圧，ピン形状，ヒートシンクの選択により，数十種類の品種を揃えている．
- 0.8/1.0/1.2/1.5/1.8/2.0/2.5/3.3/5Vの出力電圧モデルをラインナップ．
- 28Aまでの出力電流容量で，最大125W．
- 10.2～13.8Vの入力電圧範囲．
- ON/OFF制御機能を装備．
- 出力電圧調整機能，出力センス機能をもつ．

■ デイテル（株）
価格：¥5,800～（1～24個時）
TEL：03-3779-1031　FAX：03-3779-1030

■弊誌では新製品に関するニュースリリースを募集しております．
宛先は，〒170-8461 東京都豊島区巣鴨1-14-2 Interface編集部ニュースリリース係
FAX：(03)5395-2127，E-mail：mngnews@cqpub.co.jp　（編集部）

HARD WARE

●フォトカプラ

HCPL-3180 IGBT ゲートドライブ フォトカプラ

- IGBTやパワーMOSFETなどのパワー素子の高速駆動を実現するフォトカプラ.
- 最小値2Aの出力ピーク電流で，伝達遅延時間を200nsまで短縮.
- 2部品間の伝達遅延時間差は±90nsで，高速高精度応答を実現.
- トーテムポール接続されたパワー素子を駆動する際のデッドタイムのうち，フォトカプラの要因で決定する部分を180ns以下まで短縮.
- 最大65nsのパルス幅ひずみを実現しており，高性能スイッチング電源，インバータやプラズマディスプレイなどに使われるパワー素子を大容量で高速かつ高精度に駆動することが可能.

■ アジレント・テクノロジー（株）
サンプル価格：¥280～
TEL：0120-61-1280

●データ集録モジュール

NI PXI-4204 NI PXI-4220

- データ集録ボードの中に信号調節機能を搭載.
- PXI-4220は高速でのひずみ，負荷，圧力の計測など，構造テストや航空宇宙分野のアプリケーション向けに設計．PXI-4204の入力範囲は100Vとなっており，自動車における14Vや42Vの計測，テストなど，高電圧測定，計測，テストが必要となるアプリケーションに適する．いずれも，差動型同時サンプリング入力をサポート.
- PXI-4204は，八つのアナログ入力，16ビットの分解能をもち，200kspsの速度で同時サンプリングが可能．入力範囲は±100V.

■ 日本ナショナルインスツルメンツ（株）
価格：¥288,000（NI PXI-4204）
¥259,000（NI PXI-4220）
TEL：03-5472-2970　FAX：03-5472-2977
E-mail：prjapan@ni.com

●JTAG ICE

PARTNER-Jet

- USB2.0インターフェースをサポートし，実測値最高3Mバイト/sの高速ダウンロードを実現.
- 最大18Mビットの大容量トレースメモリを搭載.
- 200MHzの高速トレースクロックに対応.
- オプションコントロールソフトウェアによって，他のCPUシリーズのデバッグが可能.
- マルチコアCPUに対応.
- 拡張インターフェースにより，ROMエミュレーションが可能.
- VLINK対応で，転送速度1Mバイト/s以上.
- リアルタイムトレース，ハードウェアブレークなど，ICEに必要な機能を装備.

■ 京都マイクロコンピュータ（株）
価格：¥198,000～
TEL：075-335-1050　FAX：075-335-1051

●ディスプレイテスタ

マルチメディア ディスプレイテスタ 3298F

- フラットパネルディスプレイの輝度，コントラスト，フリッカ，色度を1台でチェック可能なテスタ.
- カラーセンサの分解能を従来の10倍に向上することで，低輝度時のコントラスト測定の精度を大幅に向上.
- 色と輝度を同時に調整できるため，色温度調整時間を約30%短縮.
- パソコンを利用して遠隔測定を行う場合の通信手段を見直し，従来品と比較して測定時間を30%短縮.
- 液晶，プラズマディスプレイや液晶プロジェクタの製造，応用機器などに適する.

■ 横河電機（株）
価格：¥580,000（カラータイプ）
¥380,000（白黒タイプ）
TEL：0422-52-6613　FAX：0422-52-6624

●プロトタイピングシステム

Virtual Turbo Linux版

- 大規模SoCの開発向けに，既存のFPGAモジュール「LogicBench」をPCIボードに搭載した「Virtual Turbo」のLinux版.
- Linux OS上で動作する市販のシミュレータと連動でき，数百万ゲートクラスの大規模SoCの論理シミュレーションが可能.
- SoC論理やシステムアプリケーションプログラムの一部を，内部にハードウェアとして実現することで，シミュレーションやプログラム処理の実行速度を高めるハードウェアアクセラレータとして利用可能.
- PC上で動作するアプリケーションプログラムと，「LogicBench」内に格納したハードウェアの設計データを連動した，協調シミュレーションが可能.
- アプリケーションプログラムからアクセスするためのAPIを用意.
- 設計データを「LogicBench」内に格納して評価ボードに搭載することで，プロトタイピング環境を構築することが可能.

■（株）ルネサス テクノロジ
価格：¥7,200,000（Virtual Turbo 24K）
¥4,700,000（Virtual Turbo 16K）
TEL：042-320-7300

●ディジタルカメラ

DVC-1412M

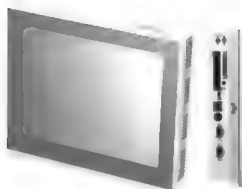
- 高分解能，高感度のディジタルカメラで，独特な2/3インチ，メガピクセルプログレッシブスキャンインターラインCCDセンサを使用.
- CCDは高変換効率と青緑にピークをもつスペクトラム感度特性なため，ほとんどのアプリケーションで利用が可能.
- リアルタイムでPCに画像取り込みが可能な「C-View」ソフトウェアを添付．カメラコントロールが組み込まれているため，画像を見ながらダイアログボックスで操作が可能.
- 5ユーザープログラマブル，シングル-クイックアプリケーション，さらにTWAINやImage-Proドライバの対応で汎用の画像処理ソフトウェアとの併用も可能.

■（株）アルゴ
価格：下記へお問い合わせ
TEL：06-6339-3366　FAX：06-6339-3365

●パネルコンピュータ

IPC-PT/LS10AC-4J
IPC-PT/LS10AC-4E
IPC-PT/MV10AC-4J
IPC-PT/MV10AC-4E

- LS10ACは12.1インチTFTカラー液晶，MV10ACは10.4インチTFTカラー液晶を搭載．
- CPUにRISCチップ「SH-4（240MHz）」，OSに「Windows CE .NET 4.2」を採用．
- XIPと同社独自技術による高速ローディングで，完全電源オフ状態から10秒以内での高速起動を実現．
- 機能拡張用としてUSBポート，10/100Base-TX，PCMCIA，RS-232-Cなど多彩なインターフェースを搭載．
- オーディオ機器内蔵で，外部ステレオスピーカ出力を標準装備．
- 128Mバイト（SDRAM）のシステムメモリ，32Mバイトのフラッシュメモリを搭載．

■（株）コンテック
価格：¥210,000～¥221,000
TEL：03-5628-9286　FAX：03-5628-9344

●USB2.0プロトコルアナライザ

LE-620HS

- 付属の解析ソフトをインストールした解析用パソコンとUSBで接続して使用．
- USB2.0/1.1規格のHigh（480Mbps），Full（12Mbps），Low（1.5Mbps）スピードに対応．
- 測定対象のUSB機器間の通信データを，回線に影響を与えることなくアナライザ本体のメモリに記録し，記録した通信ログをUSB経由で解析パソコンに転送．
- 測定対象デバイスの転送スピードは自動判定されるため設定の必要がなく，通信ログデータはパソコンのハードディスクに最大2Gバイトまで連続的に自動記録．
- 測定中でもログデータを過去に遡ってスクロール表示できるため，まれに起こる不特定条件の通信トラブルの長時間解析に有効．

■（株）ラインアイ
価格：¥688,000
TEL：075-693-0161　FAX：075-693-0163

●アナログオシロスコープ

SS-7830/SS-7805

- SS-7830は300MHz帯域で，4チャネル搭載モデル．SS-7805は50MHz帯域で，2チャネル搭載モデル．
- SS-7830は，最高100万回の高速波形更新レートにより，リアルタイム性を損なわない波形観測が可能．大振幅の信号に重畳した微少信号の観測に便利な，最大±500div相当のDCオフセット機能を搭載．FETプローブ用の電源を内蔵し，FETプローブや電流プローブをダイレクトに接続可能．
- SS-7805は，クラス最高の16kVの加速電圧をもつため，100MHzと同等の明るい波形表示を実現．

■ 岩通計測（株）
価格：¥580,000（SS-7830）
¥128,000（SS-7805）
TEL：03-5370-5474　FAX：03-5370-5492
E-mail：info-tme@iwatsu.co.jp

●ディジタルオシロスコープ

TDS7704B型

- 周波数帯域7GHz，最高サンプルレート20Gspsのディジタルフォスファオシロスコープ（DPO）．
- 立ち上がり時間43psを実現し，DPX技術による毎秒40万回以上の波形取り込みレート，最高64Mメモリ，ハードウェアクロックリカバリ3.125Gbps，トリガジッタ1.0ps，110psグリッチキャッチなどの性能を備える．
- 長時間の波形全体の中から複数の拡大表示を行い自動スクロールが可能なマルチズーム機能などを搭載．

■ 日本テクトロニクス（株）
価格：下記へ問い合わせ
TEL：03-3448-3010　FAX：0120-046-011

●フロッピディスクドライブ

FD4X-IMJ

- 4倍速タイプであるため，フロッピディスクのリード/ライトやフォーマットに要する時間を，従来の1/3～1/4に短縮．
- USBバスパワー動作であるため，ACアダプタは不要．
- USB1.1/USB2.0 フルスピード規格に準拠しているため，WindowsでもMacでも利用可能．
- Windows Me/2000/XP，MacOS 9.0以降では，ドライバのインストールは不要．
- 3モード（1.44Mバイト/1.25Mバイト/720Kバイト）でのリード/ライトに対応．

■ イメーション（株）
価格：オープン価格
TEL：03-5717-2911　FAX：03-5717-2840

●アクセスポイント

FX-DS540-APL

- IEEE802.11a準拠 54Mbps無線LAN対応の小型/軽量マイクロアクセスポイント．
- アンテナ内蔵（ダイバーシティ）の小型筐体で，場所をとらずに設置することが可能．
- 1台で最大254ステーションからの同時ログインが可能．
- 設定，管理は，Webブラウザ上で行える．FTP，Telnetなど，システムや用途に合わせて多彩なメンテナンス方法を提供．
- IPトンネル機能を搭載しているため，ルータを越えたローミング先でもネットワーク設定を変更せずに通信が可能．
- WEP（64，128，256ビット）のほか，MACアドレスフィルタリング，IEEE802.1x認証，ESSID隠し，ANY ID拒否など多彩なセキュリティ機能を搭載．

■（株）コンテック
価格：¥70,000
TEL：03-5628-9286
FAX：03-5628-9344
E-mail：
tsc@contec.co.jp

■弊誌では新製品に関するニュースリリースを募集しております．
宛先は，〒170-8461 東京都豊島区巣鴨1-14-2 Interface編集部ニュースリリース係
FAX：(03)5395-2127，E-mail：mngnews@cqpub.co.jp　（編集部）

SOFT WARE

●技術計算ソフトウェア

Mathematica 5

- 数値線形代数における，計算速度の向上．
- 疎行列（スパース行列）の高速操作に対する，広範囲なサポート．
- 常微分，偏微分方程式に対する，新世代数値ソルバの最適化．
- 複素数，実数，整数の方程式および不等式を記号的に解くための新アルゴリズムを搭載．
- 微分代数方程式のための，統合化されたソルバを搭載．
- 内点法を含む高性能最適化，および線形計画法をサポート．
- 数値ソルバにおけるベクトル関数および配列関数を幅広くサポート．
- Microsoft .NET Frameworkと完全に統合した.NET/Linkを搭載．
- DICOM，PNG，SVG，疎行列形式のインポートとエクスポートをサポート．

■ ウルフラム リサーチ アジア リミテッド
価格：¥450,000
TEL：03-3518-2880　FAX：03-3518-2877

●特許専用翻訳ソフトウェア

PAT-Transer V6 for Windows

- 実例特許対訳文を約14万セット搭載し，完全一致文はそのまま利用でき，訳の候補として類似文を表示させせることができる．
- 大容量の翻訳メモリに対応するため，高速翻訳メモリエンジンを新たに搭載．AND検索にも対応した，キーワードでの翻訳メモリの検索が可能．
- 特許公報で使われる頻度の高い語を抽出して，2万語を追加．特許用語5万語を含む特許用基本語辞書を日英65万語，英日67万語搭載．
- 2万語を搭載した自動車工学辞書を含む，22分野の専門語辞書を搭載しているため，幅広い分野での特許文の翻訳が可能．
- カタカナ表記の「ゆらぎ」にも対応するなど，日英翻訳エンジンに改良を加え，翻訳精度が向上．
- TRADOSなどの翻訳メモリをTMX形式で，トランサーの翻訳メモリとしてインポートすることが可能．ユーザーが作成したメモリをTMX形式でエクスポートすることも可能．

■（株）クロスランゲージ
価格：¥198,000，¥498,000（ネットワーク版）
TEL：03-5287-7588

●暗号メールパッケージ

CipherCraft/Mail

- 純国産次世代暗号アルゴリズム「Camellia」に対応した，暗号セキュリティソリューション．
- 暗号キーなどを毎回手作業で選択することなく，メールを自動的に暗号化できる．
- メール送信から受信までメールそのものを暗号化し，セキュリティを保つ．
- 電子メールソフトに暗号化機能をプラグインする方式とは異なり，メールプロキシ方式によってメールの暗号化と復号化を自動的に行う処理方式を採用．
- 導入にあたっての電子メールソフトやメールサーバを選ばず，使い慣れた環境をそのまま利用可能．
- Camellia，PSEC-KEMのほか，次世代米国政府標準暗号のAES，3-DESなど多数の暗号に対応した暗号化ライブラリ「CipherCraft」を採用．
- サーバ側の対応OSは，Solaris8，Windows 2000/XP，Linux（予定）．

■ NTTソフトウェア（株）
価格：¥1,000,000～25,000,000
（エンタープライズ）
¥12,000（パーソナル）
TEL：045-212-7421
E-mail：ccraft-mail@cs.ntts.co.jp

●ソフトウェア開発キット

ECHONET機器向けソフトウェア開発キット

- ECHONET規格 Ver2.11準拠の通信ミドルウェアを提供し，ECHONET機器の開発が容易に可能．
- 小型で低消費電力の同社製8ビットマイコンに対応しており，開発環境「PanaX Series」を用いることで，ECHONET機器の仕様検討，ソフトウェア開発から性能評価まで一貫して行うことが可能．
- エコーネット無線モジュール（GB-E01）および電灯線通信モジュール(ZY-9000 M04)に対応．
- リアルタイム性，コンパクト性を実現したPanaX OS Series μITRON 4.0軽実装版（自動車制御用プロファイル）に対応．
- 通信ミドルウェア搭載用のミドルボード，機器アプリケーション開発用のアプリボード，電灯線モジュールを搭載可能な電源ボードから構成．
- サンプル機器オブジェクト，サンプルドライバ，サンプルアプリケーションソフトウェアなどを提供．

■ 松下電器産業（株）
価格：下記へ問い合わせ
TEL：075-951-8151
E-mail：semiconpress@scd.mei.co.jp

●データベースアクセスエンジン

DB Connector for Curl

- 複数のデータベース環境に対して，XMLをベースとしたマルチデータベースアクセスプラットホームを提供．
- リッチクライアント環境を構築することができ，Curlで開発したアプリケーションサーバ側のプログラム開発工数ゼロを実現．
- HTTPによるアクセスを実現しているため，インターネット環境下でのマルチデータベースアクセスプラットホームの構築が可能．
- サーバ側の処理を最小限に抑えているため，大量同時セッション環境でのパフォーマンス低下を回避．
- インターネット接続環境であれば，あらゆるロケーションからデータベースとリアルタイム通信が可能で，専用のユーザ認証機能やSSL暗号化通信などのセキュリティ機能もサポート．

■（株）カール・アジアパシフィック
価格：¥2,000,000～
TEL：03-5730-3733　FAX：03-5439-2001
E-mail：sales@curlap.com

●組み込みファイルシステム

USFiles Plus

- コンパクト機器によるデータ管理のために，最適化されたパッケージ．
- コンパクトな組み込みファイルシステムとして実績があり，FAT12/16，FAT32，VFATをサポート．
- コンパクトフラッシュカード，ISO9660ファイルフォーマット（CD-ROM）に対応．
- ソニーのメモリースティックに新たに対応．
- ディジタルカメラ，PDAなどのコンシューマ機器や，FA/OA機器，医療機器などに適する．
- コンパクトなコードサイズを実現．
- サンプルドライバを多数搭載．
- アーキテクチャには依存しない．

■（株）日新システムズ
価格：下記へ問い合わせ
TEL：075-344-7800　FAX：075-344-7901
URL：http://www.co-nss.co.jp/

●GPLアセスメントサービス

GPL/Clinic

- GPLに対する，ユーザーソフトウェアの適合性を向上させるアセスメントサービス.
- 「ライセンスコンメンタール」「デザイン検証」「モジュール解析診断」「インプリメンテーション」の四つのサービスで構成される.
- 「ライセンスコンメンタール」は，GPL/LGPLについてセクションごとに，ライセンスの根拠となる関連法令を織り交ぜながら，利用についての法的な問題や頒布手法などを解説.
- 「デザイン検証」は，専用ツールを利用して，適切なモジュール分割方式あるいはコーディング方法についてアドバイスを行う.
- 「モジュール解析診断」は，専用のツールを利用して，モジュールプログラム個々の出典元やライセンス形式を解析し，GPLのライセンス条件に対する適合，不適合を診断.
- 「インプリメンテーション」は，モジュールプログラムを再検証し，GPLライセンス条件に適合するよう，モジュールプログラムの設計，実装に対するアドバイスを行う.

■（株）イーエルティ
価格：下記へ問い合わせ
TEL：03-3251-4350
URL：http://www.emblit.co.jp/

●XML統合開発環境ソフトウェア

xmlspy2004 日本語版

- Microsoft Visual Studio .NET内のエディタとして利用でき，XMLドキュメントの編集が可能.
- ファイルまたはディレクトリ間で差分情報を表示できるため，XMLドキュメントのバージョン管理などに適する.
- XPath2.0 β版に対応し，XPath1.0対応に切り替えることも可能.
- データベースとスキーマの連携機能において，Microsoft Access/ADO/ODBC，Oracleに対応.
- XML文書相互の変換用スタイルシートを自動生成する「mapforce2004 日本語版」を同時リリース．変換元および変換先のXMLスキーマを入力データとして，使用される要素タグ名を相互に関連付けることにより，変換ロジックをXSLTスタイルシートまたはJavaプログラムコードとして自動生成.

■ 東芝ITソリューション（株）
価格：¥198,000（エンタープライズ版）
¥79,800（プロフェッショナル版）
¥9,800（ホーム版）
TEL：042-340-8401　FAX：042-340-6028

●電子文書配信プラグインソフト

署名プラグイン TYPE-JS

- 銀行や公共料金などの明細書などの各種文書を，インターネット上で多数のユーザに，セキュアに配信することを可能にするプラグインソフト.
- 「Acrobat 6.0/5.0」に，認証機関が発行する電子証明書を用いた電子署名の付与や署名の検証，電子証明書の有効確認の機能を付加する.
- 文書送信側と受信側の2種類のプラグインで構成されており，受信側のプラグインは「Acrobat 6.0」上での動作が可能.
- 受信側は，文書改ざんの有無の確認や，認証機関に問い合わせせることで，文書作成者へのなりすましの有無などを確認できる.
- 認証機関が，電子署名および認証業務に関する法律の認定を受けた業務によって発行する，電子証明書の利用も可能.

■（株）日立製作所
価格：下記へお問い合わせ
TEL：03-5632-7412

●カスタムIC向け設計環境

Cadence Virtuoso custom design platform

- 高精度クラスのシリコン設計を実現する，包括的設計環境.
- カスタムICやRF，ミックスシグナルIC設計向けに開発された，スペック主導型の設計環境.
- 共通モデルと計算式を活用したマルチモードシミュレーション機能，10倍速のレイアウト機能，130nm以下のプロセスによるシリコン解析機能，プルチップ対応のミックスシグナル統合環境などを備える.
- 既存の開発環境のままで，業界標準データベース「OpenAccess」への移行が可能.
- OpenAccessとCDBA（ケイデンス製）の双方に対応.
- 高精度な寄生回路の抽出，アナログ*IR*ドロップ解析，電力系統のエレクトロマイグレーション解析などの機能を，DRC機能，LVS機能とあわせて採用.
- インダクタンスの抽出，エレクトロマイグレーション，高周波数のアナログミックスシグナル設計向けのフィールドソルバを装備.

■ 日本ケイデンス・デザイン・システムズ社
価格：下記へ問い合わせ
TEL：045-475-2221　FAX：045-475-2451

●アプリケーション開発ツール

Measurement Studio 7.0

- Microsoft Visual Studio .NET 2003完全対応の計測，テストアプリケーション開発パッケージ.
- Microsoft Visual Studio .NET完全対応のクラスライブラリとデータ集計，計測器制御プログラミングを最小限にする自動コード生成機能により，ベンダの計測器I/O統合や，ユーザーインターフェースの作成の簡素化が可能.
- 計測器I/OアシスタントとDAQアシスタント機能を搭載.
- 計測器I/Oアシスタントは，GP-IBやUSB，シリアル，VXI，その他の計測器と対話式にI/O接続が可能．自動コード生成機能により，計測器制御システムのプロトタイプの生成が可能.

■ 日本ナショナルインスツルメンツ（株）
価格：¥75,000（スタンダード版）
¥150,000（プロフェッショナル版）
¥300,000（エンタープライズ版）
TEL：03-5472-2970
E-mail：prjapan@ni.com

●組み込みLinux開発ソリューション

Platform Creation Suite Version 3.0

- 同社のボードサポートパッケージの設定と実装のためのツール.
- 個別のボード上での開発に必要なLinuxカーネル，関連するツールやドライバおよびドキュメントを含むパッケージで，同社のサイトよりダウンロード可能.
- あらかじめ移植，テストされた代表的なアーキテクチャ用のボードサポートパッケージの利用が可能.
- ARM，Coldfire，MIPS，Power PC，SHおよびその他のアーキテクチャ用のLinuxソリューションと統合されている.
- 組み込みLinux環境のプロフェッショナルレベルのツールを必要とする開発者のニーズに合わせて，開発されている.
- 自由に再配布が可能なアプリケーションの開発を可能にするビルドシステムといくつかの新しいツールを含む.

■ メトロワークス（株）
価格：下記へ問い合わせ
TEL：03-3780-6091　FAX：03-3780-6092

■ **弊誌では新製品に関するニュースリリースを募集しております.**
宛先は，〒170-8461 東京都豊島区巣鴨1-14-2 Interface編集部ニュースリリース係
FAX：(03)5395-2127，E-mail：mngnews@cqpub.co.jp　（編集部）

読者の広場

2003年10月号特集「詳説マイクロプロセッサー—パイプラインとスーパースカラ」に関して

▷ヘネパタ本を読みこなすのは重たい気がしていたところに，今回の特集が掲載されてたいへんうれしかったです．とくに最近は，改めてコンピュータ技術の基本を勉強しなおしているところです．インテル4004が登場してから，30年以上がたちましたが，あまりにも複雑になったCPUにだんだん自分自身がついていけなくなりつつあり，焦っています．（白石 隆）

▷今年に入ってから，再びファームウェア開発の仕事をはじめた．今回の特集は再びハードウェアのほうに目を向け始めていたので，おもしろい企画でした．本職はソフトウェアのほうなので，いずれソフトウェア開発技法や人工知能の応用といったテーマを取り上げてください．（スーパースカラ波）

［編］ソフトウェア開発技法の記事に期待されている方には，今月号の特集はどうだったでしょうか．

>マイクロプロセッサについての特集が良かったです．初めて購入しましたが，今後も読んでいこうと思います．（yoshiken）

Interface全般に関して

▷「シニアエンジニアの技術草子」で“買う価値のある雑誌”ということでしたが，私もなるほどと思います．雑誌編集者のみなさんには酷な言い方ですが，インターネットなどによりこれまでのような雑誌の内容や形態では，読者も満足しないのではないでしょうか？（五出のタマ）

▷「フジワラヒロタツの現場検証」が最終回となり，さびしさを感じます．時にはニヤリとし，時にはオドロカされ，時には対抗心に火をつけられ……と，気がつけば毎月買うInterfaceの最初に読む記事となっていました．細かな内容は覚えていなくても，「あった」ことは確実に残る思います．連載ありがとうございました．（ろじまるみん）

［編］「フジワラヒロタツの現場検証」の連載終了を惜しむ声は他にも多数いただいております．筆者の藤原氏には，ゆっくりと充電期間をとっていただき，機会があれば，またペンを握っていただければと考えております．

▷DesignWaveMagazineの2003年10月号の付録企画と連動した，Interface独自の記事を出されたらどうでしょうか．この基板や回路技術は，日本再生のためのキャパシティをもっている気がします．（JR9JUK）

アンケートの結果

興味のあった記事（2003年10月号で実施）

① 第3章 パイプライン処理の実際
② 第2章 パイプライン処理の概念
③ Appendix 1 エミュレーション機能の基礎
④ 第1章 プロセッサの基礎知識
⑤ プロローグ マイクロプロセッサの歴史
⑥ 第4章 並列処理の基本とスーパースカラ
⑦ 第5章 スーパースカラの実際
⑧ Appendix 2 低消費電力技術の原理
⑨ TOPPERSで学ぶRTOS技術（第2回）
⑩ フジワラヒロタツの現場検証（第72回）
⑪ 初級ドライバ開発者のためのWindowsデバイスドライバ開発テクニック（第1回）
⑫ SDIOカード開発入門（第1回）
⑬ 開発環境探訪（第22回）
⑭ ACPIによるPC/ATの電力管理とコンフィグレーション（後編）
⑮ ハッカーの常識的見聞録（第34回）
⑯ シニアエンジニアの技術草子（参拾弐之段）
⑰ メモリプロファイリングツールを開発する——実践編

特集担当デスクから

☆組み込みシステム関連の仕事をされている本誌読者は多いと思います．さまざまな開発方法論があり，最近までよく耳にするキーワードの一つが「オブジェクト指向」だったわけですが，メディアなどで取り上げられるほど現場に普及していない，という話もときどき出ます．

☆本特集は，「そもそも組み込み機器って何?」という話から始め，どんな方法論が開発効率向上につながるか「分析」し，「体系的な再利用」という要素に着目して開発することのメリット/デメリットを，具体例を使って解説しています．ただ，本文でも言及されているとおり，組み込み開発がすべからくこれでうまくいくわけではなく，数多ある方法論の中で有効かどうかを見きわめながら読んでほしい，というスタンスです．

☆エピローグにあるとおり，「特集プロジェクトメンバ」は，「100万行超の組み込みソフトに対しオブジェクト指向設計をトライする現役ソフトエンジニア，数々の難プロジェクトを品質保証の立場から助ける現場主義のテストのプロ，組み込みの問題を科学で解決するコンサルタント兼ソフト技術のソムリエ的コーディネータ，RTOSも自作する組み込み実装スペシャリスト，組み込みソフトウェアシステム分析者の卵のプロジェクトリーダ」の5人です．過去の何度かの開発手法に関する特集とくらべて特徴的なことの一つは，上記のような出自の方が総力で「組み込み開発手法」に取り組んだ結果/出力が，今回の特集の元となっていることです．

読者の広場

⑱ やり直しのための信号数学(第18回)
⑲ 開発技術者のためのアセンブラ入門(第21回)
⑳ Engineering Life in Silicon Valley(対談編)
㉑ 第6回組込みシステム開発技術展ESEC

特集『詳説マイクロプロセッサ——パイプラインとスーパースカラ』についてのアンケートの結果

Q1　CISCプロセッサを搭載したボードの開発や，その上で動くソフトウェアの開発をしたことがありますか？

①ある(54%)
②ない(46%)

Q2　RISCプロセッサを搭載したボードの開発や，その上で動くソフトウェアの開発をしたことがありますか？

①ある(30%)
②ない(70%)

(Q1とQ2の両方で“ある”と答えたかたに質問です)

Q3　CISCとRISCで差を感じたことはありますか？ それはどのような点ですか？

①ない(50%)
②ある(50%)
- キャッシュ/パイプライン/スーパースカラによる違い
- エミュレーション方式の違い
- オブジェクトベースのデバッグをするとき

Q4　プロセッサに関して，どのような記事を希望しますか？

- メジャーなプロセッサの性能比較
- ソフトウェア開発の入門から実践
- 省電力技術およびそれを配慮したソフトウェア開発

など

読者プレゼント

●**応募方法：本誌読者アンケートはがきに必要事項を記入のうえ，2003年11月30日(必着)までにご応募ください．なお当選者の発表は発送をもってかえさせていただきます．**

(1)　コンピュータはどれほど賢いのか
(5名)
舩本昇竜 著
ISBN4-88399-292-6 (株)すばる舎

Interface 次号予告

2004年1月号は11月25日発売です

『PCI & PCI-Xの徹底活用技法』

マスタ/ターゲット/イニシエータ/トランザクション/バースト転送/スプリットトランザクション

サーバ用途のPCではPCI-Xが標準で搭載されるようになってきた．またSHやMIPS，PowerPCなどの組み込み向けCPUでもPCIバスインターフェースコントローラが内蔵されるようになり，組み込み機器においてもシステムバスとしてPCIを採用する機器も増えてきている．とくに，PC環境と同じ周辺コントローラの採用を考えると，PCIバスへの接続を想定したデバイスが多いため，組み込み機器もPCIバスを採用せざるを得なくなる．

そこで次の特集では，PCIおよびPCI-Xの基礎知識について述べた後，FPGAによるPCI-X対応のハードウェア設計と，それに対応したWindows用ドライバの作成事例を解説する．また組み込み機器へのPCIバス実装時における注意点や，組み込み機器向けPCI BIOS，PCI拡張BIOSの作成方法などについても解説する．

★次号には，記事関連ファイルなどが満載されたCD-ROM『InterGigaNo.32』が付属します！

編集後記

■使っていた腕時計，何だか表示が変，と思っていた．しばらくして，長針短針は正しく動いているけれど，文字盤が回ってしまっているのに気付いた．購入した量販店にもっていったら，「買ったほうが安いです」と言われた．しばし検討の結果，「電池寿命8年」とうたっているものを購入．やっぱり変な感じが残るけれど．（洋）

■会社のPCはリース契約なわけですが，このたび更新で新しいPCが届きました．デュアルチャネルDDR333，Hyper-Threading対応の2.6"C"GHzのマシンです．振り返ると，前はPentiumIII 600MHz，その前はP5 133MHz，さらに前は20MHz前後の486．リース期間を4年とすると……ん〜ムーアの法則を若干下回ってるかな？（笑）（M）

■最近，趣味のプログラミングを復活．「書くだけなら割と簡単」なので，体力バカの私はモリモリとコードを書いているのですが，バグが少なくてメンテがしやすいもの，となるとこれが難しいわけでして．さらに再利用も視野に入れるとなると，新たな手法の導入も必要か……．今月号の特集で勉強します．（み）

■今月は映画を2本見てきました．1本目は「ロボコン」．一般視聴者的な視点から見れば面白いのでしょうけど，経験者的にはもっと製作過程の試行錯誤なんかも描いて欲しかったと，ちょっと消化不良気味．もう1本は「座頭市」．邦画ってイマイチな感じがしますけど，北野監督の座頭市は面白かったです．まさに「最強！」．（@）

■PCの普及が一段落したので，次の主戦場であるホームPCに注目が移っている．一時はゲーム機がその中心になると思われたが，今はテレビが中心になると考えられているようだ．マイクロソフト社はいろいろ手を打っているがうまくいっているようには見えないので，日本企業にも十分チャンスはありそうだ．5年後はどうなっているだろうか．（Y）

■連日，地震のニュースが流れています．関東圏に住んでいるので長い間おきていない大地震がいつくるのかと心配．通勤時間が長めの私としては，地震がおきた時にできれば自宅にいたいとは思うのですが，そうそうこちらの都合どおりにはいくわけもなく……．TVの地震関係の特番をみてしまうこの頃の私です．（Y2）

■長年の目標をついにやり遂げました．パチパチ．それは，月間の電話料金ゼロというもの．もともと電話で長話とか好きじゃないし，最近はメールや携帯で用を済ませ自宅の電話はもっぱら受信用，FAX用となっている．今月の請求書を見て心の中でガッツポーズ，イエーイ．今後もがんばろう．（太陽熱）

■私の幼少期，TVで「コンピューターおばあちゃん」という童謡がよく流れていた．当時，一般家庭に縁のなかった「コンピューター」と縁側でお茶をすする「おばあちゃん」を童謡で融合したその感覚は，時代の先を行きすぎていたのかもしれない……．（な）

お知らせ

▶読者の広場

本誌に関するご意見・ご希望などを，綴じ込みのハガキでお寄せください．読者の広場への掲載分には粗品を進呈いたします．なお，掲載に際しては表現の一部を変更させていただくことがありますので，あらかじめご了承ください．

▶投稿歓迎

本誌に投稿をご希望の方は，連絡先（自宅/勤務先）を明記のうえ，テーマ，内容の概要をレポート用紙1〜2枚にまとめて「Interface投稿係」までご送付ください．メールでお送りいただいても結構です（送り先はsupportinter@cqpub.co.jpまで）．追って採否をお知らせいたします．なお，採用分には小社規定の原稿料をお支払いいたします．

▶本誌掲載記事についてのご注意

本誌掲載記事には著作権があり，示されている技術には工業所有権が確立されている場合があります．したがって，個人で利用される場合以外は，所有者の許諾が必要です．また，掲載された回路，技術，プログラムなどを利用して生じたトラブルについては，小社ならびに著作権者は責任を負いかねますので，ご了承ください．

本誌掲載記事をCQ出版（株）の承諾なしに，書籍，雑誌，Webといった媒体の形態を問わず，転載，複写することを禁じます．

▶コピーサービスのご案内

本誌バックナンバーの掲載記事については，在庫（原則として24か月分）のないものに限りコピーサービスを行っています．コピー体裁は雑誌見開きの，複写機による白黒コピーです．なお，コピーの発送には多少時間がかかる場合があります．

●コピー料金（税込み）
1ページにつき100円

●発送手数料（判型に関わらず）
1〜10ページ：100円，11〜30ページ：200円，31〜50ページ：300円，51〜100ページ：400円，101ページ以上：600円

●送付金額の算出方法
総ページ数×100円＋発送手数料

●入金方法
現金書留か郵便小為替による郵送

●明記事項
雑誌名，年月号，記事タイトル，開始ページ，総ページ数

●宛て先
〒170-8461　東京都豊島区巣鴨1-14-2
CQ出版株式会社　コピーサービス係
（TEL：03-5395-4211，FAX：03-5395-1642）

▶お問い合わせ先のご案内

●在庫，バックナンバー，年間購読送付先変更に関して
販売部：03-5395-2141

●広告に関して
広告部：03-5395-2133

●雑誌本文に関して
編集部：03-5395-2122

記事内容に関するご質問は，返信用封筒を同封して編集部宛てに郵送してくださるようお願いいたします．筆者に回送してお答えいたします．

Interface

©CQ出版（株）　2003　振替　00100-7-10665
2003年12月号　第29巻　第12号（通巻第318号）
2003年12月1日発行（毎月1日発行）
定価は裏表紙に表示してあります

発行人／増田久喜
編集人／相原　洋
編集／大野典宏　村上真紀　山口光樹　落合幸喜　小林由美子
デザイン・DTP／クニメディア株式会社
表紙デザイン／株式会社プランニング・ロケッツ
本文イラスト／唐沢睦子
広告／澤辺　彰　中元正夫　菅原利江

発行所／CQ出版株式会社　〒170-8461　東京都豊島区巣鴨1-14-2
電話／編集部（03）5395-2122　URL http://www.cqpub.co.jp/interface/
広告部（03）5395-2133　インターフェース編集部へのメール
販売部（03）5395-2141　supportinter@cqpub.co.jp
CQ Publishing Co.,Ltd.／1-14-2 Sugamo, Toshima-ku, Tokyo 170-8461, Japan
印刷／クニメディア株式会社　美和印刷株式会社
製本／星野製本株式会社

日本ABC協会加盟誌
（新聞雑誌部数公査機構）　ISSN0387-9569

本書に記載されている社名，および製品名は，一般に開発メーカーの登録商標または商標です．なお本文中では™，®，©の各表示を明記しておりません．

Printed in Japan

本誌は有名書店ほか，コンビニエンス・ストア「セブン-イレブン」でも予約購読ができます．詳しくは，お店にお問い合わせください．